C 3.212:48

technical paper

48

TOPOLOGICAL PRINCIPLES IN CARTOGRAPHY

U.S. Department of Commerce
BUREAU OF THE CENSUS

Digitized by the Internet Archive
in 2012 with funding from
LYRASIS Members and Sloan Foundation

http://archive.org/details/topologicalprinc00corb

technical paper

48

Issued December 1979

TOPOLOGICAL PRINCIPLES IN CARTOGRAPHY

by James P. Corbett

U.S. Department of Commerce
Luther H. Hodges, Jr.,
Under Secretary
Courtenay M. Slater,
Chief Economist

BUREAU OF THE CENSUS
Vincent P. Barabba,
Director

BUREAU OF THE CENSUS

Vincent P. Barabba, Director
Daniel B. Levine, Deputy Director
Barbara Bailar, Associate Director for
Statistical Standards and Methodology

STATISTICAL RESEARCH DIVISION
James L. O'Brien, Acting Chief

ACKNOWLEDGEMENTS

Support for the writing of this monograph was initiated by **Harold Nisselson**, Chief of the Statistical Research Division of the Bureau of the Census. Valuable administrative support was provided by James O 'Brien, Assistant Division Chief.

The mathematical staff of the Division provided many of the examples which illustrate the text. Special recognition is due **Marvin White**, who read the text many times over, and contributed valuable suggestions, carried forward the development of the cartographic system ARITHMICON in accord with the spirit of this monograph.

Editorial functions were performed by **Linda Ambill**, Publications Services Division.

Library of Congress Cataloging in Publication Data

Corbett, James P
Topological principles in cartography.

(Technical paper - Bureau of the Census ; 48)
1. Geography, Mathematical. 2. Combinatorial topology. 3. Cartography—Data processing.
I. Title. II. Series: United States. Bureau of the Census. Technical paper ; 48.
GA23.C67 526 **79-25596**

For sale by Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402, or any U.S. Department of Commerce district office. Postage stamps not acceptable; currency submitted at sender's risk. Remittances from foreign countries must be by international money order or by a draft on a U.S. bank.
Stock Number 003-024-02132-3

FOREWORD

I am particularly pleased to introduce this monograph, which sets forth the mathematical theory underlying the applied research in cartography carried out at the Bureau of the Census. During my tenure as Director of the Bureau, I have consistently stressed the need for high quality graphic presentations, especially cartographic presentations.

There is abundant evidence that computers can provide greatly improved graphic presentation techniques, but these improvements can only be built on a sound theoretical foundation.

The topological theory presented here is the foundation for the Bureau's GBF/DIME program and for continued Bureau research. It is a mathematical basis upon which computer science can build a fully automated cartographic system.

During the International Symposium on Computer-Assisted Cartography in Reston, Virginia, September 23, 1975, I challenged the participants to advance the state of the art more in the coming decade than our predecessors have accomplished in the last century. I hope that this publication will be not only a guide to practitioners of automated cartography, but also a stimulus to researchers in meeting the challenge.

The Bureau welcomes comments and suggestions from users of this technical paper.

VINCENT P. BARABBA
Director
Bureau of the Census

Contents

	Page
Part I. Topological Principles in Cartography	1
Introduction	1
Combinatorial topology	1
Maps	1
Cells	2
Oriented Cells	2
Complexes	3
Chains	3
Boundary and coboundary operators	3
Circuits, or cycles	4
Manifolds	4
Orientable manifolds	4
Fusion and subdivision of cells	5
Fusion	5
Map of a closed two-dimensional manifold	6
Identification	7
The DIME representation of a segment—duality	7
Neighborhoods	8
Fundamental neighborhood of a O-cell	9
Linear graphs	10
Topology of a map	11
Psuedo-dual graphs	12
Boundary of a cartographic area	15
Metrical and topological inconsistencies	15
Folds	16
Neighborhoods on a Disk	16
Representative graph codes	17
Graph analysis	18
Map editing	22
Representation of chains and operators	24
Part II. ARITHMICON: A System of Computerized Cartography	30
System objectives	30
System components and functions	30
The subsystem SPEAK	31
The SPEAK language	31
The SPEAK stack system	33
The SPEAK sybsystem	37
The subsystem TOPO	41
Editing procedures	42
The subsystem GRAPH (components and functions)	44
Summary of ARITHMICON procedures	47
APPENDIXES	
Appendix A. Homology	49
Appendix B. Syntax of SPEAK	50

Part I. Topological Principles in Cartography

INTRODUCTION

The essential feature of a system of computerized cartography is the use of an arithmetic model of a map. From an information theoretic point of view, a properly designed model would be equivalent to the source maps which the model is intended to represent. The most primitive model of this kind is the simple tracing in which the graphic structure on the map is represented by polygonal arcs defined by the coordinates which determine the endpoints of the arcs. If such a model contains, in addition to the metrical descriptions of the graphic structure, the textual annotation, properly located with respect to the graphic elements, then the question of the equivalence of the model to the source map is merely a question of the fidelity of the tracing and the degree of approximation attained by the method used for representing the polygonal arcs of the graph.

A number of projects have been carried out that have as their objective the production of maps by this method or some closely related method. Technical problems connected with attaining the desired fidelity of representation aside, there are some important practical difficulties associated with this method. These difficulties manifest themselves in two ways: first, severe optical clutter occurs attending attempts to plot from erroneous model data, and second, difficulties are encountered in attempting to retrieve data linked to the map.

The difficulties are largely eliminated if the model of the map is extended to contain an explicit description of the cellular structure of the map. The cellular structure of a map is defined by the graphic elements which represent linear arcs, their terminal points, and the areas bounded by polygonal chains of arcs. Throughout the remainder of this monograph, such cellular objects will be denoted as 0-cells, 1-cells, and 2-cells respectively. The first advantage of such a model is that the cells are the natural cartographic objects with which data having geographic linkages may be made to correspond. Every possible finitary geometric structure associated with a map must be susceptible to description in this way. The second advantage is one that is often seriously underestimated. A cellular model may be manipulated algebraically. As a consequence, the geometric constraints inherent to a cell structure may be used as tests of the consistency of the model, both in terms of the properties of individual cells, and in terms of the global relational scheme. Either of these advantages standing alone would more than justify the adoption of a cellular model as an extension of simple tracing. Taken together they provide more than ample justification.

The cell structure simplifies the problems associated with data retrieval. When retrieving data by direct visual inspection of actual maps, one encounters serious problems in localizing selected cartographic objects. An automated system provides for automatic searching of indexes, and selective plotting of specified cartographic structures. Thus, the data retrieval problem is completely and efficiently automated by the cellular model.

The fact that the consistency of the cellular model may be tested by simple algebraic procedures is the basis of a comprehensive scheme of editing which virtually assures the geometric self-consistency, not only of the map model, but of the source map from which it was derived. The major objective of this monograph is to provide explicit technical documentation from which such a system may be reconstructed. To this end the topological relations must be expressed as a formal system of relations which may be reduced to a computerized representation or model.

Equally important is the presentation of a computerized realization of this formal system. The particular realization is the system of automated cartography known as ARITHMICON. The extant version of this system is written in MACRO-10, for use in the DEC-10 computer.

This documentation is being designed to serve two purposes. First, it serves as a guide to the interpretation of the present realization of the system ARITHMICON. Second, it is intended to serve as a reference for the use of those who may wish to construct an independent system of computerized cartography for their own purposes.

COMBINATORIAL TOPOLOGY

In this section, the main facts of combinatorial topology related to practical cartography are presented descriptively. The material presented here is taken directly from standard texts, and represents a paraphrase of descriptions from these sources, together with a commentary which is interpolated for clarifying the usage of certain technical terms.

Maps

From a topological point of view, a cartographic map is defined as a mapping function from ordinary Euclidean three-space to itself. A mapping that is pointwise one-to-one (bi-unique), continuous, and having a continuous inverse (bi-continuous) is known as a homeomorphism. All of the usual cartographic mappings, Mercator, transverse Mercator, polyconic, and similar mappings, are examples of homeomorphic mappings.

Cells

The elementary objects of combinatorial topology are cells.

The Euclidean n-sphere is defined as the point set

$$\Sigma x_i^2 = 1; \quad i = 1, \ldots, n$$

An open n-cell is any homeomorph of the interior of this sphere, the set of points

$$\Sigma x_i^2 < 1; \quad i = 1, \ldots, n$$

A homeomorph of the closed n-sphere is called a closed n-cell. At this point some clarifying remarks are interpolated. Cells defined as closed cells are sometimes called nonsingular. Cells may be defined in such a way that their boundaries are singular, that is the mapping which defines the boundary cells may be merely continuous, and not one-to-one. Such cells will be referred to as singular cells. In figure I-1, the diagram at the left is a 2-cell with a boundary having a single singular point. In the diagram on the right, the boundary contains a singular 1-cell. One may regard each of these diagrams as a result of a continuous deformation of a regular 2-cell which brings some set of boundary points into coincidence at the singular boundary element of the deformed cell.

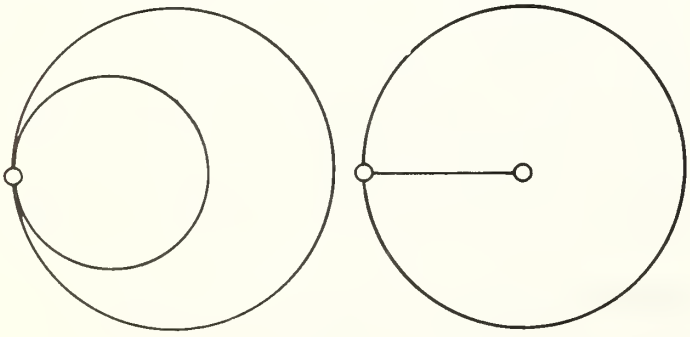


Figure I-1

Examples of homeomorphs which are 2-cells abound in ordinary geometry. The plane triangle, either as a closed or open cell, is the homeomorph of the interior and boundary of the unit circle. Any plane polygon together with its interior forms a closed 2-cell. Any closed polygon on the sphere bounds an open 2-cell on the sphere.

The homeomorphisms defining cells induce relations of incidence between cells and their boundaries. These relations will be represented as ordered pairs of cell identifiers:

$$(c^0, c^1) \qquad (1)$$

for 0-cells and 1-cells, and

$$(c^1, c^2) \qquad (2)$$

for 1-cells and 2-cells. The convention adopted here places the cell of lower dimension first in the pair.

Oriented Cells

Two orientations may be established for any cell. Formally, an oriented cell is distinguished from an unoriented cell by associating with the cell identifier an algebraic sign (+, −). A 0-cell is oriented arbitrarily. A 1-cell is oriented by choosing one of the two impossible orderings of its boundary 0-cells. A 2-cell is oriented by selecting one of the two possible cyclic orderings of three distinct points on its boundary. Formally, then, a cell requires a triple for its complete representation:

(dimension, sign, identifier) (3)

In the formal representations adopted in ARITHMICON, these three elements may be represented implicitly, as in the following example. The pairs

(−1, 1)
(2, 1)

imply only relative dimension, and may serve equally to represent a relation between (0, 1) or (1, 2) dimensional pairs. In the first case, the pairs represent the 1-cell 1 and its boundary 0-cells −1 and 2. In the second case, the pairs serve to represent two 1-cells on the boundary of a given 2-cell. The elements of an incident pair will be known respectively as

(boundary, coboundary) (4)

In figure I-2, a 2-cell with three 1-cells and three 0-cells on its boundary is shown. The list of incident pairs of ordered cells is displayed in association with the figure.

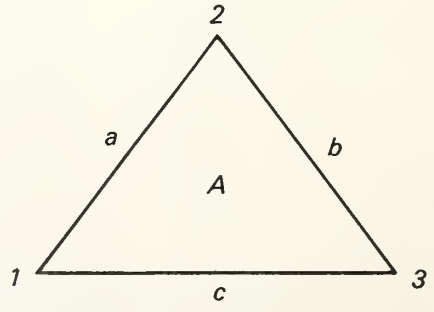


Figure I-2

The associated incident pairs are

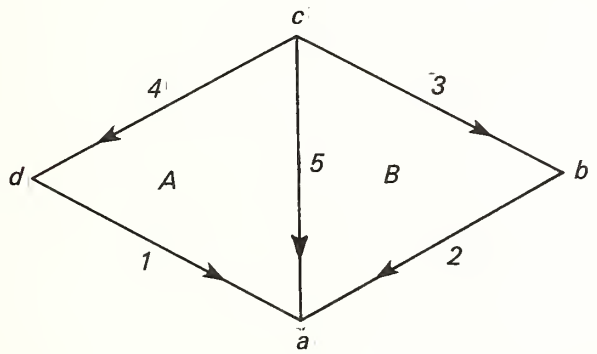
(−1, a), (2, a), (−2, b), (3, b), (−3, c), (1, c) ; (0-cells, 1-cells)
(a, A), (b, A), (c, A) ; (1-cells, 2-cells)

Complexes

A complex is a cellular structure formed according to the following set of recursive rules:

- A zero-dimensional complex (a 0-complex) is a collection of 0-cells.
- A one-dimensional complex (a 1-complex) consists of a 0-complex (known as the 0-skeleton) together with a set of 1-cells satisfying the conditions: Each 1-cell is bounded in the 0-skeleton, and each 0-cell of the skeleton is on the boundary of some 1-cell of the complex.
- A two-dimensional complex (a 2-complex) is a 1-complex (known as the 1-skeleton) together with a collection of 2-cells satisfying the conditions: Each 2-cell is bounded in the 1-skeleton, and each 1-cell of the skeleton is on the boundary of some 2-cell of the complex.

An illustration of a simple 2-complex and its representation in terms of incident pairs is given in figure I-3.



(0, 1) dimension: (a, 1), (a, 5), (a, 2), (—c, 5), (—c, 4), (—c, 3), (—d, 1), (d, 4), (—b, 2), (b, 3)

(1, 2) dimension: (4, A), (1, A), (—5, A), (5, B), (—2, B), (—3, B)

Figure I-3

It is simple to verify that the structure in the figure above satisfies the conditions of a complex. The 0-skeleton is the set of four 0-cells,

(a, b, c, d)

and each 1-cell of the set (1, 2, 3, 4, 5) has its boundary points in this 0-skeleton. Each 0-point is on the boundary of one of the 1-cells of the set. The pair of 2-cells is each bounded in the 1-skeleton, and each 1-cell of the skeleton is on the boundary of one of the 2-cells.

These relations imposed on a cellular complex form the basis of the first step in editing a map model. As indicated, the verification of the complex rule is a simple algebraic procedure.

Chains

A chain is a collection of cells of the same dimension. A chain may be represented by a set of symbols in the following way. If the collection is, for example, the set of 1-cells, selected from the set of n_1 1-cells,

$$C = (c_1, \ldots, c_m) \quad ; m \leqslant n_1 \tag{5}$$

then the chain, C, is represented as a vector,

$$C = (x_1, \ldots, x_{n_1}) \quad ; n_i = \text{Number of i-ceils} \tag{6}$$

The x_i are either all signed integers or all integers mod 2.

This principle obviously applies to chains of any dimension, and is obviously equivalent to representing a chain as its characteristic function over the set of all cells of the specified dimension.

Chain addition is carried out component by component. In the case of nonoriented cells the addition is interpreted mod 2. In the oriented case the addition is interpreted as addition of signed integers.

Boundary and Coboundary Operators

It can be seen from the representation of the incidence relations among oriented and nonoriented cells that the relation for the case of nonsingular cells can be represented by its characteristic function. In the nonoriented case, this characteristic function is defined to be equal to one for pairs that are incident, and zero otherwise. For oriented cells, the characteristic function takes on three values, zero for pairs not incident, one for pairs having the same orientation, and minus one for pairs having opposite orientation. It will suffice at this point to ignore the case in which there are singular cells, since the actual representation used in ARITHMICON will not be a conventional matrix representation.

Whatever form the representation of the incidence relations may take the relations will be denoted by a set of four operations, E_0^1, E_1^0, E_2^1, and E_1^2. It will suffice to define the last of these:

The operator, E_1^2 operating on a cell or on a cell chain of dimension 1, is defined to be the collection of 2-cells incident with any of the cells of the chain.

Thus

$$E_1^2 C^1 \tag{7}$$

is the coboundary of the chain C^1.

The remaining operators may be defined by merely changing the appropriate dimension numbers in the foregoing definition. In this definition it is understood that the boundary or coboundary chains thus formed are added according to the appropriate mode of addition of cell chains as previously described.

The following example will sufficiently explain the interpretation of these operators. In figure I-4,

$$E_2^1\ (1) = (1, 1, 1) \qquad (8)$$

$$E_0^1\ (1, 0, 1) = (1, 0, 1) + (1, 1, 0) = (0, 1, 1) \qquad (9)$$

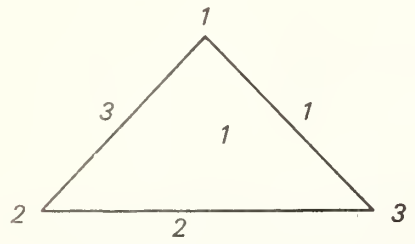


Figure I-4

Circuits, or Cycles

The terms "circuit" and "cycle" are used synonymously. A cell chain is a circuit if its boundary is null and if no subchain has this property. Any chain having a null boundary is either a circuit or a set of circuits. For example, the boundary of a nonsingular 2-cell is a 1-chain that is a 1-circuit.

The terms "cocircuit" and "cocycle" are also used synonymously. The definition is analogous to that of circuit. A 1-chain is a 1-cocircuit if its coboundary is null and if no subchain has this property.

The following examples illustrate these concepts. The illustration in figure I-5 represents a tetrahedron. First consider the cells to be nonoriented.

$$E_0^1(1, 1, 1, 1) = (0, 0, 0, 1, 1, 1) + (1, 0, 1, 0, 1, 0) + \ldots$$
$$\ldots (0, 1, 1, 1, 0, 0) + (1, 1, 0, 0, 0, 1) =$$
$$(0, 0, 0, 0, 0, 0)$$

$$E_1^0(1, 0, 0, 0, 1, 1) = (0, 1, 0, 1) + (0, 1, 1, 0) + (0, 0, 1, 1)$$
$$= (0, 0, 0, 0)$$

$$E_2^1(1, 1, 1, 1) = (1, 1, 1, 0, 0, 0) + (0, 1, 0, 1, 0, 1) +$$
$$\ldots (0, 0, 1, 1, 1, 0) + (1, 0, 0, 0, 1, 1) =$$
$$(0, 0, 0, 0, 0)$$

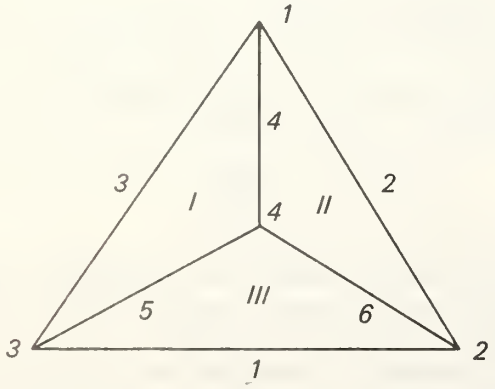


Figure I-5

Manifolds

A two-dimensional manifold is a 2-circuit which, in addition, satisfies the condition that each point has a neighborhood homeomorphic to a flat disk. The conditions stated imply that—

1. Each 1-cell of a cellular manifold is incident with exactly two 2-cells.
2. At each 0-cell, the incident 1-cells and abutting 2-cells form a chain of alternating 1-cells and 2-cells, and this chain is a simple cyclic chain. This condition will be expressed by stating that the coboundary of each 0-cell is a cocircuit.

Manifolds defined in this way are called closed manifolds. The term closed is used in the same sense as that intended when we speak of closed surfaces, such as the sphere. The closed manifold has no one-dimensional boundary.

A geometric object obtained by removing disjoint 2-cells from a closed manifold is known as an open manifold. For strict regularity it is required that the boundary of this manifold consist of nonintersecting 1-circuits. An open manifold can be transformed into a unique closed manifold by replacing the removed disks.

The tetrahedron, figure I-5, is a two-dimensional manifold. Its boundary is null, as may be verified by the computation illustrated. There is a countably infinite set of nonhomeomorphic two-dimensional manifolds. Common examples are the sphere, the torus, the Klein bottle, and the projective plane. These are each closed manifolds. Examples of nonhomeomorphic open manifolds are the Mobius strip, the disk, and the annular ring.

Orientable Manifolds

The concept of orientation has already been defined for cells. An oriented two-dimensional manifold is a manifold for which a cell orientation may be chosen so that each 1-cell is oriented once positively and once negatively with respect to the pair of incident 2-cells.

There is a very simple algorithm for determining whether or not a given manifold, specified by a collection of incident pairs of cells, is or is not orientable.

First, choose a 2-cell arbitrarily. To establish a fixed convention, orient the 1-cells on the boundary in such a way that when the boundary is traversed in the counter-clockwise direction, the interior of the bounded 2-cell lies to the left. Each 1-cell of the bounding 1-circuit is assigned a positive orientation.

Next, choose any 2-cell having 1-cells in common with the first chosen 2-cell. The 1-circuit bounding the newly chosen 2-cell is now oriented in such a way that the already oriented 1-cells are negatively oriented with respect to this 1-circuit. Such a choice may not be possible. In this case, the manifold is not orientable.

If the second chosen cell can be consistently oriented with respect to the negatively oriented 1-cells, the remaining 1-cells

of the bounding 1-circuit are oriented positively with respect to this circuit.

The procedure continues, choosing successive 2-cells having boundary 1-cells in common with the already oriented 1-cells. If every 2-cell can be oriented to satisfy the conditions stated in the preceding paragraph, the manifold is orientable.

Fusion and Subdivision of Cells

A nonsingular 1-cell has distinct bounding 0-cells. If a third 0-cell is chosen on the 1-cell, it is necessarily between the bounding 0-cells of the original 1-cell. Removal of this cell from the 1-cell leaves a pair of 1-cells having a common boundary 0-cell. By continuing to subdivide the cells thus obtained, the original 1-cell may be partitioned into an arbitrarily large number of 1-cells and boundary 0-cells.

Counting the number of 1-cells and 0-cells of the original nonsingular 1-cell there are two 0-cells and one 1-cell. A formula connecting these numbers defines an invariant for a one-dimensional complex:

$$\chi^1 = n_0 - n_1 \tag{10}$$

This number is known as the characteristic of the one-dimensional complex.

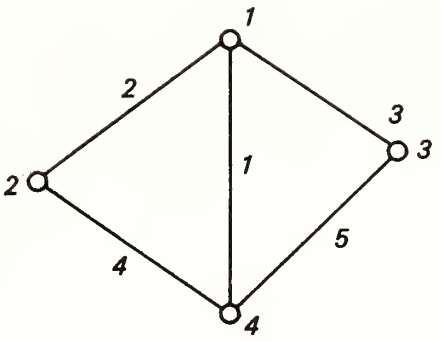
The characteristic defined above is invariant under the process of subdivision described in the first paragraph above. Each chosen 0-cell on the complex, when removed, leaves a pair of 1-cells replacing the subdivided 1-cell. The number of 0-cells and the number of 1-cells each is increased by one.

For a closed one-dimensional manifold, or 1-circuit, the characteristic has the value 0. For a simple 1-cell, the characteristic has value 1. Beginning with a 1-cell, which has characteristic value 1, and adding 1-cells by attaching one end to one of the 0-cells of the boundary, and adding the necessary 0-cells to complete the boundaries of the attached 1-cells, it can be seen that the characteristic value is unaltered by this process. Since the 1-cells attached are each accompanied by one boundary 0-cell, the 1-cells and 0-cells are added in pairs. A structure formed in this way is known as a tree.

On the other hand, suppose one begins with a tree, which, as has just been shown, has characteristic value 1. Adding to this complex a single 1-cell, having both bounding 0-cells already in the complex, reduces the characteristic value by 1. Each such added 1-cell equally reduces the characteristic value. Since each such added 1-cell forms a closed loop independent of any previously formed loop, one may determine the number of closed loops or 1-circuits simply by counting 0-cells and 1-cells. Thus, the value of the characteristic is zero for a 1-complex containing a single circuit. Denoting the number of independent 1-circuits by μ,

$$\mu = 1 - \chi^1 \tag{11}$$

Figure I-6, illustrates the determination of the number of 1-circuits of a 1-complex or linear graph.



$\chi^1 = 4 - 5 = -1$

$\mu = 1 - (-1) = 2$

Figure I-6

We next turn to the partition of 2-cells. Any 2-cell with a nonsingular boundary may be partitioned by a 1-cell having its boundary 0-cells on the boundary of the 2-cell, and itself interior to the 2-cell. The invariant characteristic for two-dimensional complexes is

$$\chi^2 = n_0 - n_1 + n_2 \tag{12}$$

The partition of a 2-cell by a 1-cell as described above increases the number of 2-cells and the number of 1-cells, each by unity. The insertion of the boundary 0-cells of the introduced 1-cell onto the boundary of the 2-cell has no effect on the characteristic. The characteristic of the two-dimensional complex is therefore invariant under partitions.

Fusion

Any 0-cell on a 1-complex which is incident to exactly two 1-cells may be fused with these 1-cells to form a 1-cell. This procedure of fusion is essentially inverse to the procedure of partition of such a cell. Obviously, the characteristic value is unaltered by this operation. In the case of 2-cells, any pair of 2-cells which share a common 1-cell on their boundaries may be fused, provided that their boundaries have no other points in common, and of course, neither of the 2-cells has an interior point in common. Such a fusion results in a new 2-cell which is the union of the common 1-cell and its incident 2-cells. Obviously, the removal of the 1-cell and the associated reduction in the number of 2-cells by one, leaves the characteristic value of the complex unaltered. Figure I-7 illustrates the fusion of a pair of 2-cells.

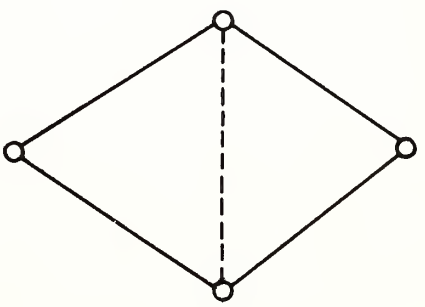


Figure I-7

The preceding paragraphs show how the number of 1-circuits of a two-dimensional complex may be determined from the characteristic of the 1-skeleton. From the fact that each 2-cell has a bounding 1-circuit in the 1-skeleton, the number of independent bounding 1-circuits cannot exceed n_2. But the number of independent 1-circuits bounding 2-cells cannot exceed the number of 2-cells less one, since the fact that the two-dimensional manifold is a 2-circuit means that there is a linear relation holding on the set of boundary 1-circuits of these cells; in fact, their sum (mod 2) is zero. There are therefore exactly $n_2 - 1$ independent 1-circuits bounding the 2-cells of the manifold. The number of nonbounding 1-circuits may therefore be determined by subtraction. By convention, the number of nonbounding 1-circuits is known as

$$R_1 - 1 = \mu - (n_2 - 1) = 2 - \chi^1 - n_2 \text{; from which}$$

$$R_1 = 3 - \chi^2 \qquad (13)$$

For an open two-dimensional manifold a closely related formula may be obtained by a slight modification of this method. Since for an open manifold some 2-cell must have been removed from a closed manifold, it follows that the remaining 2-cells must have linearly independent boundary 1-circuits. The number of bounding 1-circuits is therefore equal to n_2. It follows from this that the value of R_1 for an open two-dimensional manifold is given by the formula

$$R_1 = 2 - \chi^2 \qquad (14)$$

The fundamental theorem of the combinatorial topology of the closed two-dimensional manifold states that any two manifolds, both orientable, or both nonorientable, and having the same characteristic are homeomorphic.

Map of a Closed Two-Dimensional Manifold

In standard texts on combinatorial topology, the following fundamental facts about 2-manifolds are shown. Every closed 2-manifold can be set into one-to-one correspondence with the points of a convex polygon in the plane, in such a way that each interior point of the polygon is the correspondent of a single point of the manifold, and the boundary points which correspond to the edges of the polygon may be made to correspond by pairs, and such pairs each correspond to a single point of the manifold, and finally the vertices of the polygon as a set are each the correspondent of one and the same point of the manifold. The number of edges of the polygon is equal to twice the number of nonbounding 1-circuits of the manifold.

This theorem is important in computerized cartography because it illustrates the way in which an anomalous manifold may be inadvertently constructed by faulty identification in a model. For the purpose of cartographic interpretation, this theorem is best explained by citing examples.

The first example, shown in figure I-8a, is a representation of the torus, or anchor ring.

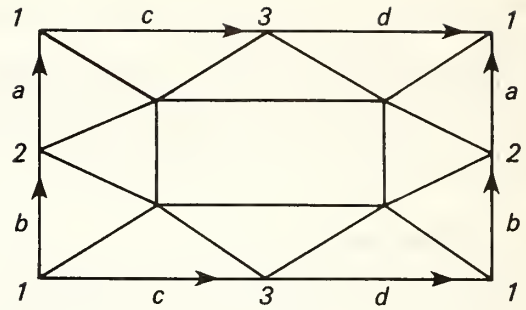


Figure I-8a

Figure I-8a represents a triangulation of the torus, and will be analyzed according to the principles already explained in the text. The first part of the procedure is to determine whether the manifold represented is or is not orientable. This example may be made clearer by the introduction of a closely related manifold, having a representation differing only slightly from that of the torus, namely the Klein bottle. This is shown in figure I-8b.

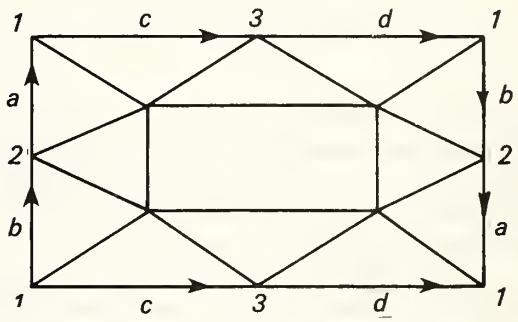


Figure I-8b

Note that in each case the triangulation of the rectangle is identical. The only difference between the figures is in the identification of the edges on the right and left of the figure. Since the interior of the rectangle is known to be orientable, it remains only to consider the segments on the edges. Thus for figure I-8a, the edges are oriented once positively and once negatively relative to a 2-cell interior to the rectangle, that is in one case there is a positively oriented 2-cell on the left, and again on the right of the segment. This being true for the 1-cells on the boundary, and necessarily true for the 1-cells interior to the rectangle, the manifold represented is orientable. On the other hand the cells a and b cannot be so oriented in figure I-8b, since the cells a and b as shown have positively oriented cells on the left hand only when oriented as indicated in the diagram. The Klein bottle is therefore not an orientable manifold.

Each of these manifolds has the same characteristic, 0, and the number of independent nonbounding 1-circuits is in each case 2. The essential difference is the nonorientability of the Klein bottle, and the orientability of the torus.

Identification

The last examples illustrate the principle of identification. A difference in the identifications of cells is capable of producing a radical alteration in the interpretation of a diagram. This phenomenon is the source of a relatively large number of errors in coded maps.

The most common form of misidentification is that in which two or more 0-cells are identified under a common identifier. The topological structure induced by this identification is illustrated in figure I-9.

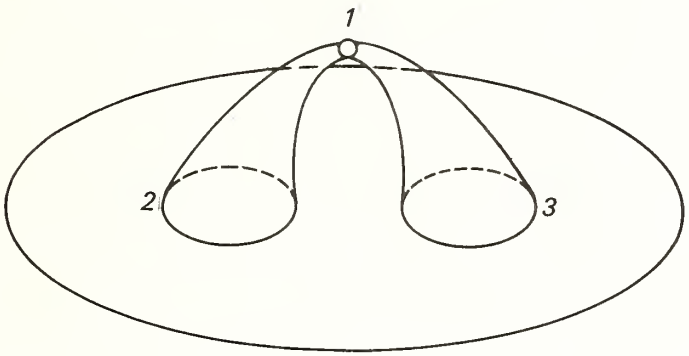


Figure I-9

The complex obtained in such a case is not strictly a manifold, since it fails to have a flat two-dimensional neighborhood at the point 1. The neighborhood consists of a pair of disks, joined at a common point. A simple map can be drawn by modifying the map of the annular ring, as shown in figure I-10.

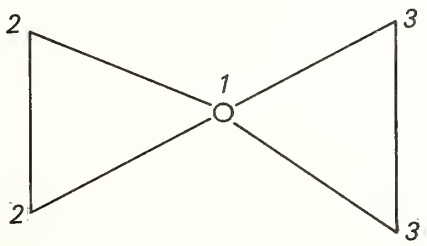


Figure I-10

When this singular manifold is joined to a disk with two holes, one obtains the equivalent structure shown in figure I-9.

It will be seen that there is no possibility for misidentification of segments, in the sense that two distinct geometric segments are given a common identifier. There is, however, the possibility that geometrically distinct 2-cells are given a common identifier. The means of detecting such an anomaly will be described in the section "Topology of Maps."

A commonly occurring error is one in which the boundary segments of a 2-cell are traced in the wrong order. This leads to an interchange of the identification of a pair of 0-cells on the boundary, and a Mobius strip is inadvertently created. This kind of structure is illustrated in figure I-11.

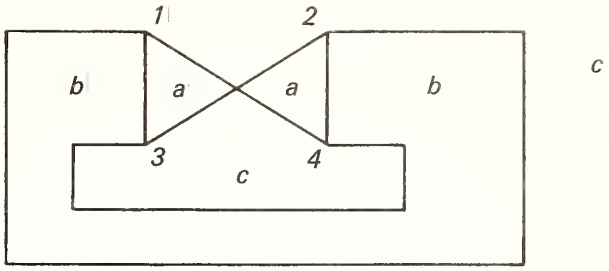


Figure I-11

The cellular structure shown should be an annular ring, but has been converted to a Mobius strip by the misidentification created at the time of tracing. If the domains adjacent to the ring were identified, neither would have a boundary, but their union would be bounded by a simple circuit, the same 1-circuit that bounds the Mobius strip. The entire structure is therefore equivalent to a projective plane.

The DIME Representation of a Segment—Duality

A two-dimensional manifold is characterized by an important form of symmetry in its incidence structure. This form of symmetry is known as duality. Pairs of incident cells have already been denoted as paired elements in a relational scheme, (relative, correlative), as in (boundary, coboundary). Boundary and coboundary are dual aspects of one and the same incidence relation. Cells are also subject to a duality relation. On a two-dimensional manifold, a 0-cell is a dual to a 2-cell, and 1-cells are dual to 1-cells. Thus, given a valid incidence scheme representing a cellular structure on a two-dimensional manifold, there is a dual structure defined by exchanging relative and correlatives in the incidence relations, while interpreting cellular objects as dual cellular objects entering into these incidence relations. Because of this geometric fact, it is possible to represent each 1-cell of a cellular structure of this kind by means of a quintruple

(s, a, b, c, d)

satisfying the dimensional schema

(1, 0, 0, 2, 2)

and geometrically interpreted as a directed 1-cell associated with an orientation schema

(s, From, To, Right, Left)

which in terms of oriented cells is equivalent to a sign schema

(+, –, +, –, +)

A simple example will illustrate this principle. Figure I-12 shows the map of the tetrahedron. The DIME representation of the incidence relations is indicated in the figure.

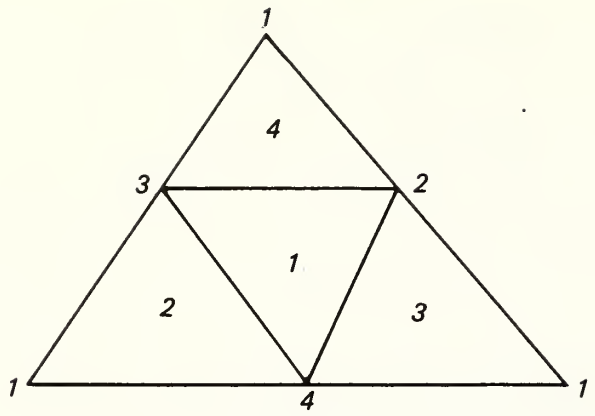


Figure I-12

DIME segments

(1, 3, 4, 1, 2)
(2, 2, 4, 3, 1)
(3, 1, 4, 2, 3)
(4, 3, 1, 2, 4)
(5, 2, 3, 1, 4)
(6, 1, 2, 3, 4)

Dual DIME segments

(1*, 1, 2, 4, 3)
(2*, 3, 1, 4, 2)
(3*, 2, 3, 4, 1)
(4*, 2, 4, 1, 3)
(5*, 1, 4, 3, 2)
(6*, 3, 4, 2, 1)

In this case, the structure is self-dual; that is, there is a 1-1 correspondence between cells of the original complex and cells of the dual which preserves incidence relations.

The segments s and the transformed segments s* represent just one form of the representation of a complex on a manifold. A second version is obtained by choosing opposite orientations of the 1-cells. This representation is expressed by the transformation

$$(s, a, b, c, d) \rightarrow (s', b, a, d, c) \qquad (15)$$

The opposite choice of orientation of a 1-cell can be interpreted geometrically as viewing the manifold from the opposite side. A similar interpretation may be made of negatively oriented 0-cells. This convention permits the representation of a nonorientable manifolds by means of a DIME code. The representation of the projective plane in figure I-13 is an example.

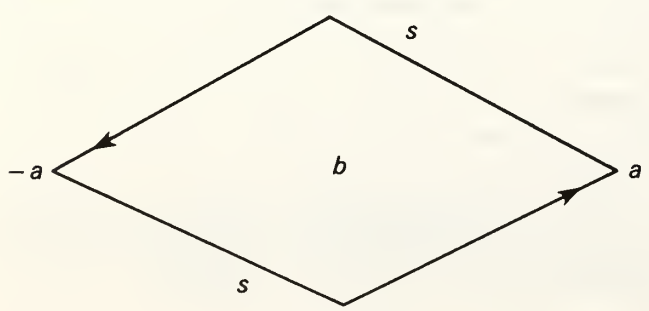


Figure I-13

The DIME code for this manifold consists of the single singular segment

s; (a, −a, b, −b)

When one determines the boundary of the 2-cell b, one obtains the segment s, similarly oriented, twice. The boundary is therefore 2 s. Evidently, the segment is self-dual.

Interpretation of a DIME Segment. From the dimensional scheme of a DIME segment, the fourth and fifth fields are two-dimensional objects. It will always be assumed that these objects are topologically equivalent to flat disks, or 2-cells. If it is desired to represent nonorientable objects in terms of DIME segments, this can be done only by the introduction of singular segments.

The geometric object corresponding to a two-dimensional DIME code must, in many cases, be constructed. The method of representation allows a 2-cell to contain other 2-cells in their interior, and so on to some finite degree of nesting. The DIME code identifier is always interpreted to be a complete 2-cell containing no holes, or alternatively containing holes which are filled by other identified 2-cells.

Although the 2-cells thus defined are orientable open manifolds, it does not follow that the structures formed by fusion of 2-cells will themselves be orientable. One may encounter coding errors which wrongly imply that a map structure formed of fused 2-cells is not orientable. Thus a test for orientability is required.

A procedure for obtaining the complete 2-cell as a fusion which replaces holes is provided among the examples of data retrieval.

Neighborhoods

The basic editing procedures are directed toward the verification of the fact that each vertex has a fundamental neighborhood homeomorphic to a 2-cell, if the vertex is interior to the manifold, and a neighborhood homeomorphic to a hemi-disk, if the vertex is on a boundary of the manifold.

In interpreting DIME segments related to a given area code, the code itself is always interpreted as an identifier of a flat two-dimensional manifold, that is, as a disk, or a disk with holes. Since such a structure can be made to correspond in a unique way to a disk, simply by replacing the holes by simple open disks, all of these structures may be regarded as simple disks without any loss of generality. It is only required that the structure internal to any such disk be strictly accounted for.

While this interpretation may be made for the fundamental areas distinguished in the model, the result of such an interpretation is not necessarily a consistent geometric object. The consistency is established only through the process of applying the constraints that the cellular object resulting from such an interpretation of the model satisfies the conditions on a complex, and in addition, the conditions on a manifold.

Any encoded fundamental area must satisfy the condition that it is a disk or that it is a disk with holes. In turn, the holes

themselves must represent encoded areas. To simplify matters for the purpose of discussion, the map itself may be considered to be a disk. In this case, the fundamental neighborhood of any 0-cell must itself be a disk. In case the disk has holes, the areas corresponding to the interior of these holes must be analyzed.

The steps in building up a fundamental neighborhood of a vertex begin with the identification of the individual cells which, when fused together, form the neighborhood. In the section dealing with retrieval of neighborhood cells, the method of assembling these cells was illustrated. The illustrations given were purposely simplified. Actually, one must allow for the presence of holes in the neighborhoods assembled by the given operator sequences. This requires that the internal loops of the neighborhood boundary be identified. The coboundaries of these neighborhoods must be adjoined to the collection of 2-cells to be fused. These adjoined 2-cells, may themselves have holes. The finite character of the model assures us that the procedure of restoring the removed 2-cells to the neighborhood will finally terminate. The end result should be the assembly of a simple disk.

What is assumed about the interpretation of an area code as an identifier of a 2-cell is justified by the procedure described above. The 2-cell mentioned in a DIME segment refers to the 2-cell obtained after restoring any and all of the removed holes. Figure I-29 illustrates the types of internal holes that may be encountered. For a discussion of the procedure for retrieving the list of 2-cells interior to a given 1-circuit bounding a disk, see the example on the use of the procedure, HOMOL, and the discussion of homology, p. 27.

Fundamental Neighborhood of a 0-Cell

The sequence of operations resulting in the collection of the cells forming a fundamental neighborhood of a 0-cell is now presented. Beginning with the 0-cell, c^0,

$$C^1 = E_0^1 c^0 \tag{16}$$

This one-dimensional coboundary is shown in figure I-14.

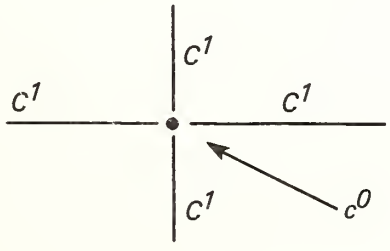


Figure I-14

The two-dimensional coboundary of the 1-chain

$$C^2 = E_1^2 C^1 \tag{17}$$

is shown in figure I-15.

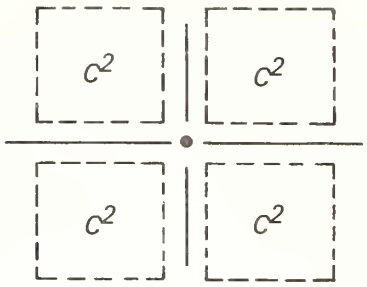


Figure I-15

These two operations identify the collection of cells whose union forms the open neighborhood of the 0-cell. This is the smallest open neighborhood of the vertex definable in terms of the cell structure. It should be noted that this system of neighborhoods covers the entire cell structure, but that individual points are not necessarily in one-to-one correspondence with the neighborhoods of the system. One and the same neighborhood may be the fundamental neighborhood of a finite number of 0-cells.

The closures of these neighborhoods are obtained by adjoining the point sets defined by the following operators:

$$B^1 = E_2^1 C^2 \tag{18}$$

$$B^0 = E_1^0 B^1 \tag{19}$$

The closed neighborhood is shown in figure I-16.

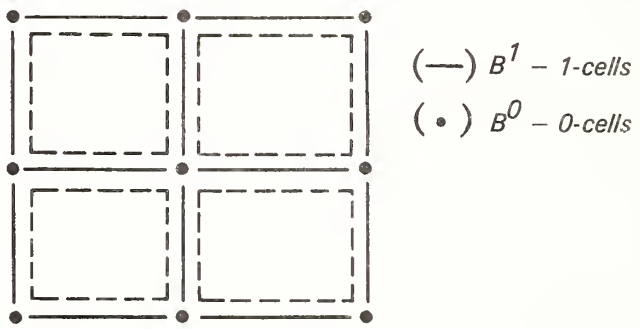


Figure I-16

The next neighborhood to be constructed is the closed neighborhood of a 2-cell, and the smallest open neighborhood containing this closure. Beginning with the 2-cell, the following operator sequence will assemble the cells making up the closed neighborhood:

$$B^1 = E_2^1 C^2 \tag{20}$$

$$B^0 = E_1^0 B^1 \tag{21}$$

The union of these last two sets, together with the original 2-cell, constitutes the closed 2-cell, the smallest closed neighborhood containing the cell itself. This neighborhood is represented by the closed central rectangle in figure I-17.

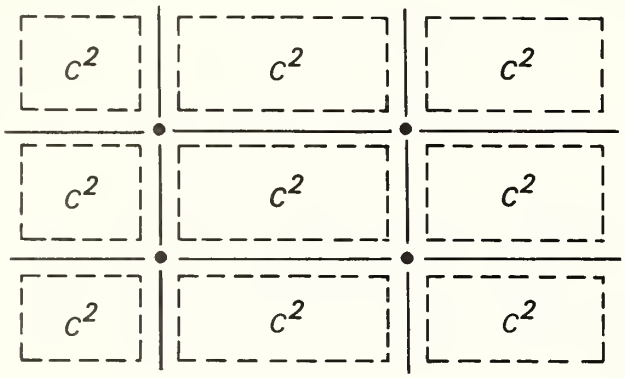


Figure I-17

The smallest open neighborhood containing the closed neighborhood of a 2-cell is obtained by the following operators:

$$C^1 = E_0^1 B^0 \tag{18}$$

$$C^2 = E_1^2 C^1 \tag{19}$$

The neighborhood obtained by adjoining these point sets to the closure of a 2-cell is the smallest open set containing the closed 2-cell. Note that the cellular structure of the closure of the fundamental neighborhood of a 0-cell is precisely dual to the structure of the open neighborhood covering a 2-cell and its boundary.

The examples of neighborhoods shown above have been purposely chosen to be as simple as possible. Some idea of the complexity of a neighborhood will be presented in the section on the topology of maps. It is unnecessary here to give a special example of the fundamental neighborhood of a 1-cell, since this neighborhood is simply the union of the fundamental neighborhoods of the pair of bounding 0-cells.

Linear Graphs

A one-dimensional complex is also known as a linear graph. A wide diversity of terminology exists in connection with graphs. The 0-cells of the complex are known as vertices, nodes, or endpoints. The 1-cells of a graph are known as segments, edges, or arcs. A graph which contains an arc connecting each pair of 0-cells is said to be connected.

For representations of 0-chains, mod 2, a 0-circuit consists of a pair of 0-cells. By convention the boundary of such a pair is the sum, mod 2, of the coefficients, which in this case each equal 1. For representations using signed integers, a 0-circuit consists of a pair of 0-cells of opposite orientation. Thus, in either case the boundary is zero or null. The sum of any number of 0-circuits is a 0-circuit.

A 0-circuit is a bounding 0-circuit if it is the boundary of a 1-chain, and all other 0-circuits are nonbounding. For a graph that is connected, there are no nonbounding 0-circuits. If a graph is not connected, the maximal connected sets of cells are known as components. The number of components is related to the number of independent nonbounding 0-circuits. The number of independent 0-circuits is denoted by R_0. There are $R_0 - 1$ components in a linear graph.

The relation between the characteristic of a connected graph and the number of independent 1-circuits is specified in equation (11). This equation is applied separately to each component of the graph. Although we can determine the number of independent 1-circuits from the two characteristics, R_0 and R_1, further analysis is required to identify the 1-cells not in any proper 1-circuit, the so-called acyclic segments.

In order to simplify the analysis of graphs, each graph will be first transformed by eliminating any 0-cells or vertices of index 2. Such a graph will be said to be reduced. This transformation is identically that already discussed under the subject of the fusion of 1-cells having a single boundary 0-cell common to their boundaries. Since this transformation does not affect the characteristic of the graph, the result is to provide a graph which together with the known condition of being reduced, has invariant numbers of 0-cells and 1-cells.

For the purposes of analytic cartography, the aim of this graph analysis is to provide a concise characterization of the graph. The method decided upon is to specify for each graph a numerical code consisting of the following sequence of integers:

$$c; (n_1, n_0, \mu, \alpha) \tag{20}$$

The term μ is redundant in this list but is included to save an editor from making the calculation of its value; α is the number of acyclic segments. The symbol c denotes the particular component. Figure I-18 illustrates a graph from an actual urban map and its graph code.

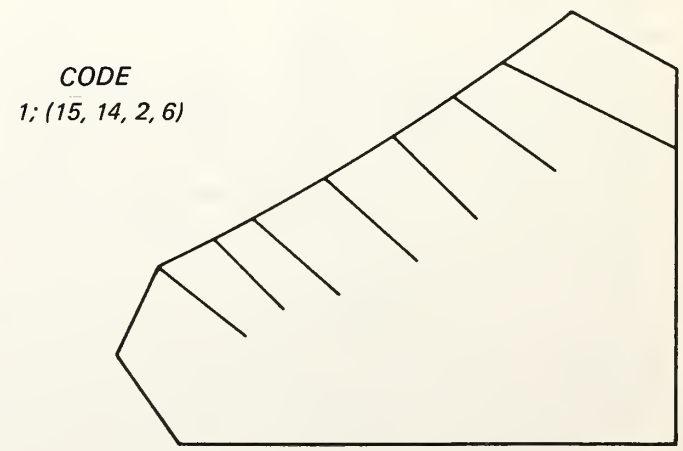


Figure I-18

The next example, figure I-19, illustrates the 1-skeleton of a fundamental neighborhood. This example is included because of the fact that the analysis of such a graph is a fundamental part of every edit of a 0-cell or vertex neighborhood.

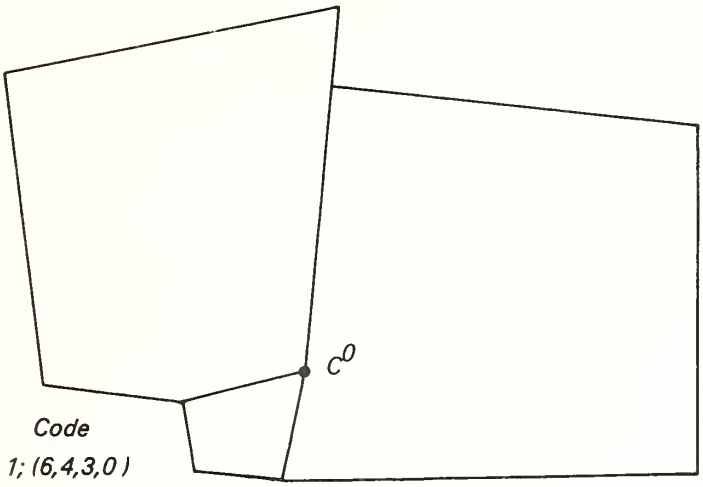


Figure I-19

An important structure that will be frequently encountered in the analysis of a neighborhood is the disk with holes. Figure I-20 illustrates a simple case and the associated graph code.

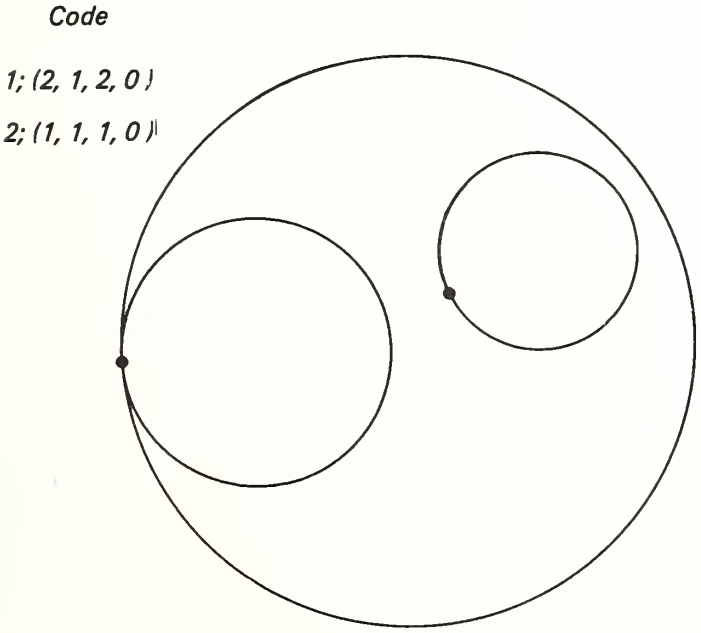


Figure I-20

A more complete example on the projective plane will be used to illustrate the method of representing a nonorientable manifold by means of a DIME code. The example is based on the structure in figure I-21 which shows a map of the Mobius strip, closed by the insertion of a disk.

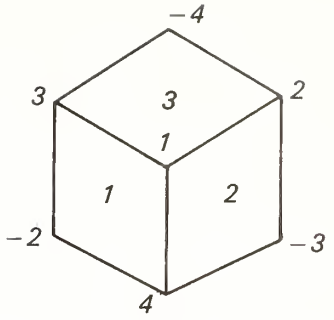


Figure I-21

In this figure, the oriented 0-cells are interpreted as points in the surface viewed from the opposite side of the surface. There are paths from a positively oriented 0-cell to its negatively oriented version on a nonoriented surface. This fact has led to nonorientable surfaces being called one-sided surfaces. The DIME code for this figure, and its dual figure, is shown below:

1; (1, 3, 1, 3)	1^*; (1, 3, 3, 1)
2; (1, 2, 3, 2)	2^*; (3, 2, 2, 1)
3; (1, 4, 2, 1)	3^*; (2, 1, 4, 1)
4; (−3, 2, 2, −1)	4^*; (2, −1, 2, −3)
5; (2, −4, 3, −1)	5^*; (3, −1, −4, 2)
6; (4, −3, 2, −3)	6^*; (2, −3, −3, 4)

The DIME segments and the dual segments can be verified by inspection of figure I-21.

TOPOLOGY OF A MAP

The present section deals with the interpretation of topological ideas in cartography. It has already been pointed out that the graphic and cellular structures encountered on actual maps are often singular. For example, the boundary of the disk, shown below, is singular, and moreover the disk itself does not conform to the definition of a manifold. It must be regarded as a singular manifold, the singularity being the double point on the boundary. At this point, there is not a neighborhood homoemorphic to a hemi-disk. On the contrary, the neighborhood is disconnected into two components by the removal of the singular point.

As has already been pointed out, cellular structures of this kind cannot be represented by the usual form of incidence matrix. However, such cells can always be subdivided, in the way already explained, so that the resulting complex can be represented by a set of incidence matrices. The example in figure I-22 shows a map composed of singular cell structures.

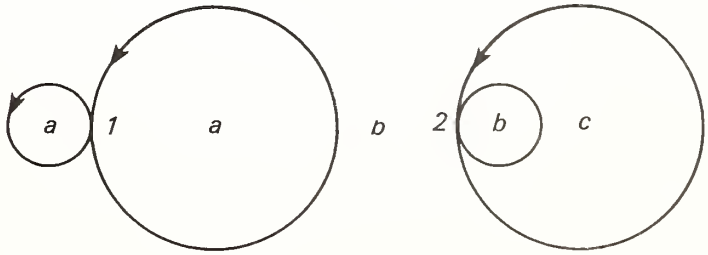


Figure I-22

The one- and two-dimensional characteristic values for the individual disks are also computed for the purpose of illustration.

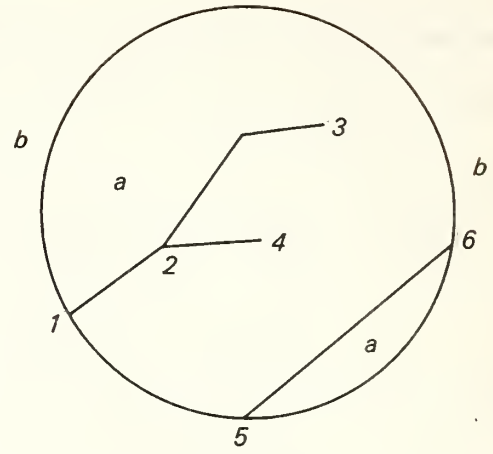
The DIME code for the map shown in figure I-22 is the following:

1: (1, 1, a, b)	3: (2, 2, c, b)
2: (1, 1, a, b)	4: (2, 2, c, b)

Because of the singularities of the cell structure, the left-hand figure represented by segments 1 and 2 is topologically indistinguishible from the right-hand figure represented by the segments 3 and 4. That is, there is a homeomorphism induced by an exchange of the label pairs (1, 2), (a, c), and (b, b). Not only are the cellular structures equivalent, but they are represented by equivalent segment pairs. Such cases can be distinguished only by use of the metrical data associated with the 1-cells. This data permits one to distinguish the right- and left-hand figures of the map, and to see that the disk c contains a hole, b. The areas marked with the symbol b, represent a disk, with a bulge and some holes.

The two-dimensional characteristic of the disk a is equal to one. The figure may be regarded as being obtained by coalescing two disks at a point on their boundary. This reduces the number of 0-cells of any triangulation by one, and hence the characteristic of a pair of disks which equals two is reduced by unity. On the other hand, the disk c has a hole, b. The value of the characteristic is equal to zero. The figure may be regarded as the result of coalescing two boundary points of a nonsingular disk to a single point. This reduces the characteristic by unity, and gives the value zero as a result. The area b is a disk with three holes, a, a, and c, together with a bulge into the hole c. For each component of the boundary of the figure, the one-dimensional characteristic is the same, namely, minus one. Each component is therefore a pair of loops joined at a single point. These examples cover the phenomenology as far as the cyclic graphic structure is concerned. It remains to consider the acyclic structure of the graphs. Since the disks are assumed to be flat, or genus zero, every graph having circuits interior to the disk either bounds or is part of a boundary of some interior hole. On the other hand, the acyclic structure is not properly a part of the boundary.

Any tree structure internal to a disk boundary may be regarded as a part of a singular boundary, obtained by coalescing 0-cells and 1-cells of a nonsingular boundary. Such singularities are associated with corresponding singularities of the DIME segments, indicated by the presence of identical codes in the 2-cell positions of a segment. Such a singularity will be referred to as a singularity of the second kind.



Boundary segments	Interior segments
1; (1, 6, b, a)	4; (5, 6, a, a)
2; (6, 5, b, a)	5; (1, 2, a, a)
3; (5, 1, b, a)	6; (2, 3, a, a)
	7; (2, 4, a, a)

Figure I-23

Pseudo-Dual Graphs

The presence of cells with singular boundaries complicates the analysis of conditions at a vertex. For this reason, no attempt is made to construct an actual dual graph during the editing procedure. Instead, a pseudo-dual graph is drawn. The procedure for constructing this graph is illustrated in figure I-24. This illustration shows the utility of the pseudo-dual in providing a visual indication of a sense reversal of the sides of a segment. The reversal is indicated by the presence of the bow tie in what would otherwise be a simple loop.

In the actual dual as analyzed in the internal chaining procedure, the reversal is detected as a segment negatively oriented in the chain. However, the graphic indication conveys this information more directly.

Drawing a pseudo-dual in this way requires a preliminary ordering of the 1-cells at the vertex. This ordering is cyclic and is induced by the angular positions of the individual segments. Once the segments have been ordered, the chain may be drawn as long as the successive segments have one and the same 2-cell on their common coboundary. When this condition does not hold, the chain cannot be continued.

There is an obvious cell mapping from the pseudo-dual to the dual. The mapping is possibly singular.

The following example of the use of the pseudo-dual is somewhat more complex. Figure I-25 illustrates a vertex with singular segments. The original code is defective and the

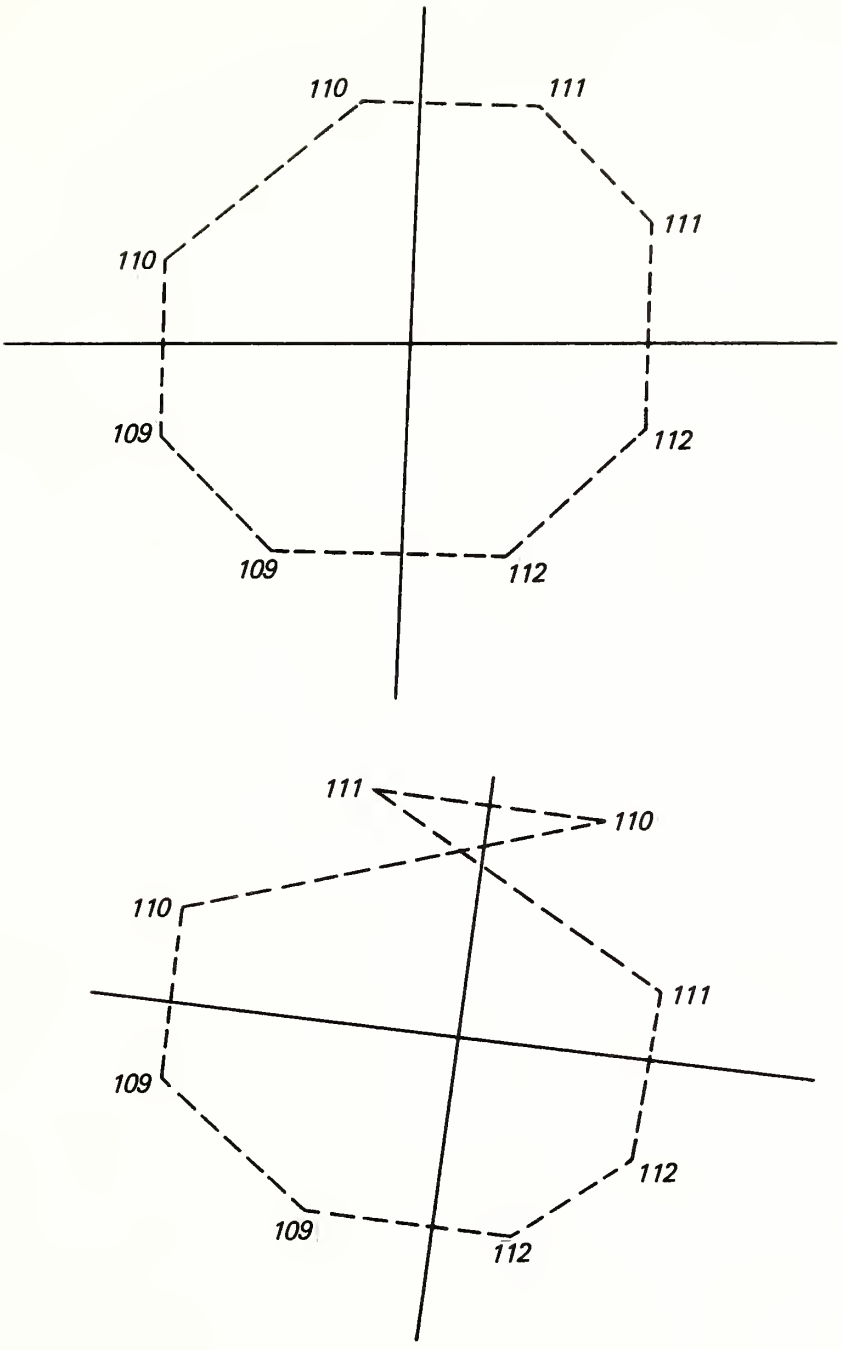


Figure I-24

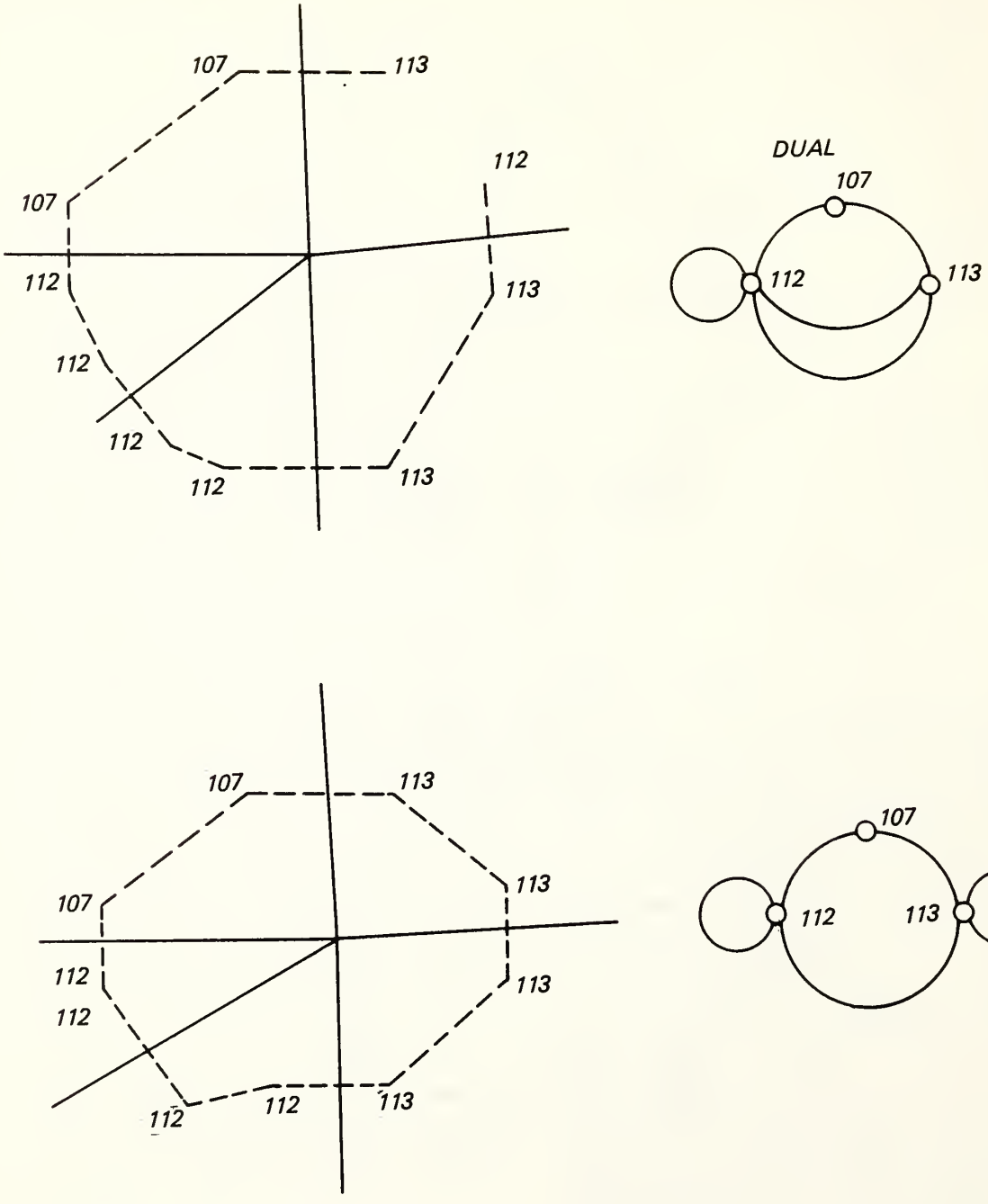

107
113
112
107
112
113
112
112
113
112
113
DUAL
107
112
113
107
113
107
113
112
112
113
112
113
107
112
113

Figure I-25

pseudo-dual consists of a single acyclic segment. The corrected version is also shown in the figure.

The difference between duals and pseudo-duals is also indicated. The associated dual is displayed adjacent to each of the pseudo-duals. By tracing the segments of the pseudo-dual on the actual dual, one may easily verify that a continuous map of the pseudo-dual may be drawn on the actual dual. It is quite evident in this case that the pseudo-dual provides a more direct indication of the location and kind of error than does the actual dual.

Boundary of a Cartographic Area

The fact that a cartographic area may be a disk with holes complicates the editing procedures. For example, figure I-26 shows the results of defective coding of a fundamental neighborhood of a vertex. As shown in the abstract version, the neighborhood should be an annular ring. In fact, the outer ring is broken and, in addition, has a spurious self-intersection. The fact that the vertex is consistently located relative to the annular ring is verified analytically by determining first that the point is interior to the outer ring, and second, that it is exterior to the inner ring. Of course, the outer and inner rings were first identified by appropriate analytic procedures.

There is of course no way of knowing that the self-intersection shown at vertex 703 is spurious. This information is the result of adjudication from the actual map.

The example, which incidentally is taken from the encoded map of Jackson, Mississippi, indicates the necessity for complete analysis of the boundaries of a fundamental region.

Metrical and Topological Inconsistencies

The coordinate descriptions of points, 1-cells, and 2-cells may not be consistent with the set of incidence relations among these cells. The most frequently occurring errors of this kind result from displacements of points as expressed in terms of their assigned coordinates to positions outside of their fundamental neighborhoods. There are two simple ways of detecting such displacements. First, a point-in-polygon test may be made of the position of the point relative to the boundary of its fundamental neighborhood. Second, a displacement may be detected by the observation of folds indicated by retrograde motion of the pseudo-dual relative to its general cyclic direction.

The first example illustrates detection of a point location exterior to the boundary of its fundamental neighborhood. Figure I-27 illustrates a case of this kind. The position of the point as originally coded is indicated by an asterisk, and the corrected position is indicated in the same way.

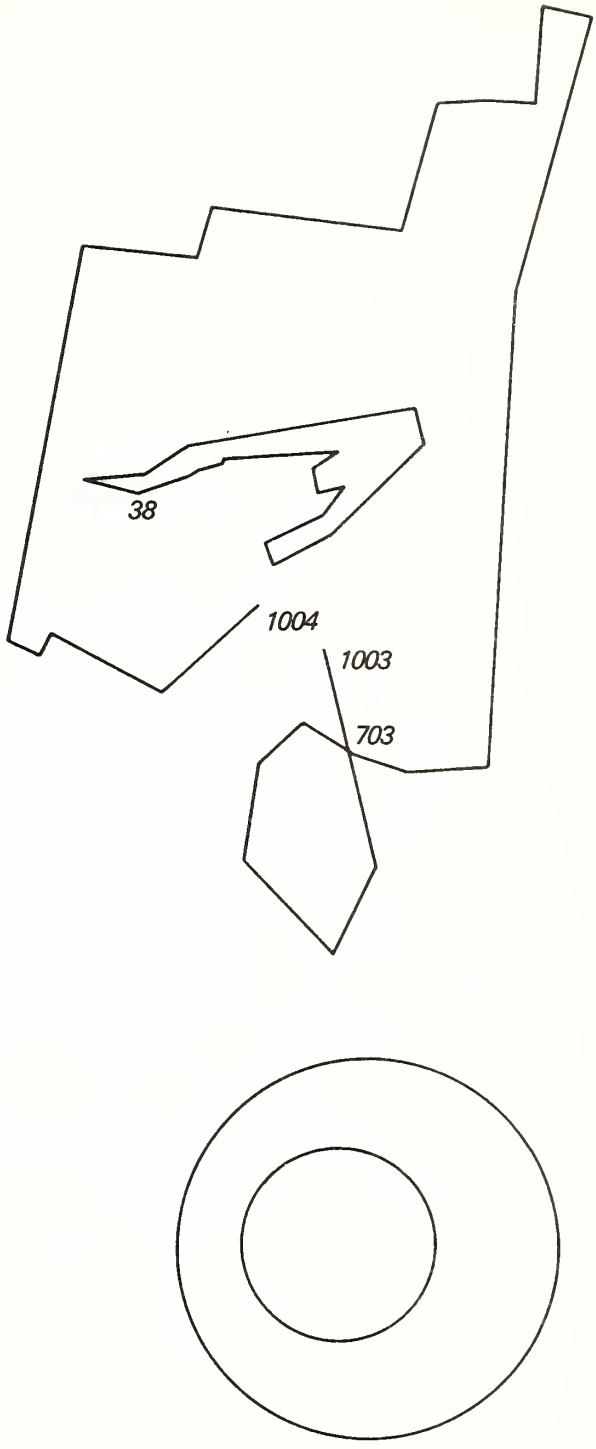
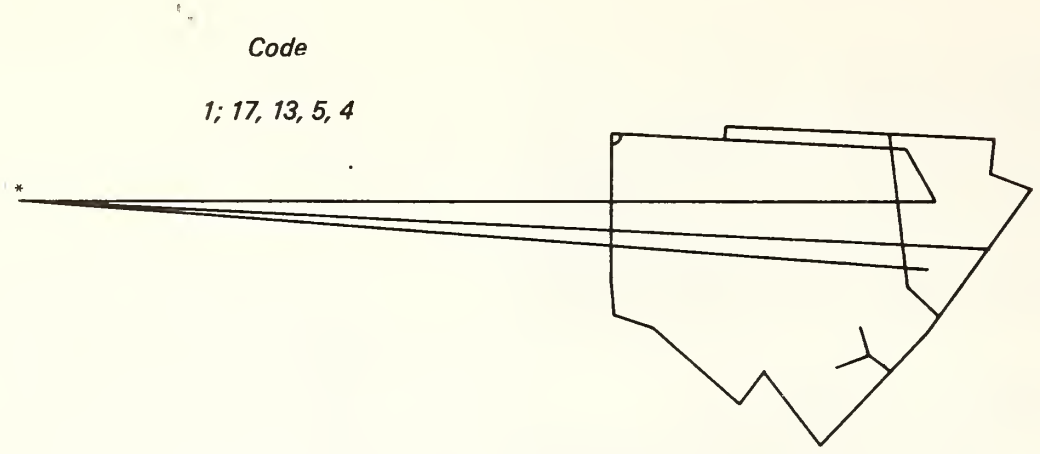
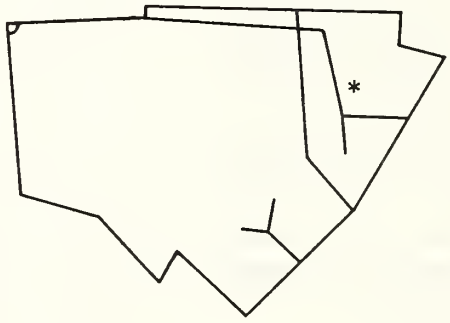


Figure I-26



Vertex displaced exterior to fundamental neighborhood



Vertex repositioned

Figure I-27

Folds

Although a drastic displacement of a vertex may be detected by the means already described, a point-in-polygon test of the location of the point relative to the boundary of its fundamental neighborhood, it may often be detected by another means. Such a displacement may cause a fold in the neighborhood of a vertex. Two examples of such folding are shown in figure I-28.

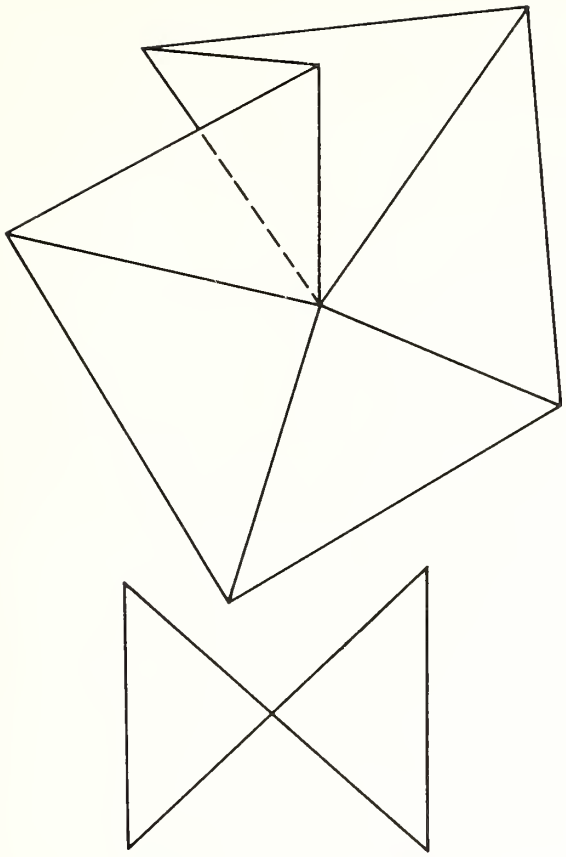
A fold of this kind will be detected by a retrograde movement of the pseudo-dual graph relative to its direction of circulation about the vertex. This movement is different from the bow tie characteristic which accompanies a side-reversal of a segment.

Such a retrograde movement occurs because of an inconsistency in the two possible ways of ordering the segments in the coboundary of the vertex. First, the segments are ordered according to their angular position at the vertex. Second, they are ordered by topological position in the coboundary chain. Inconsistency of these two orderings will produce the retrograde movement.

Neighborhoods on a Disk

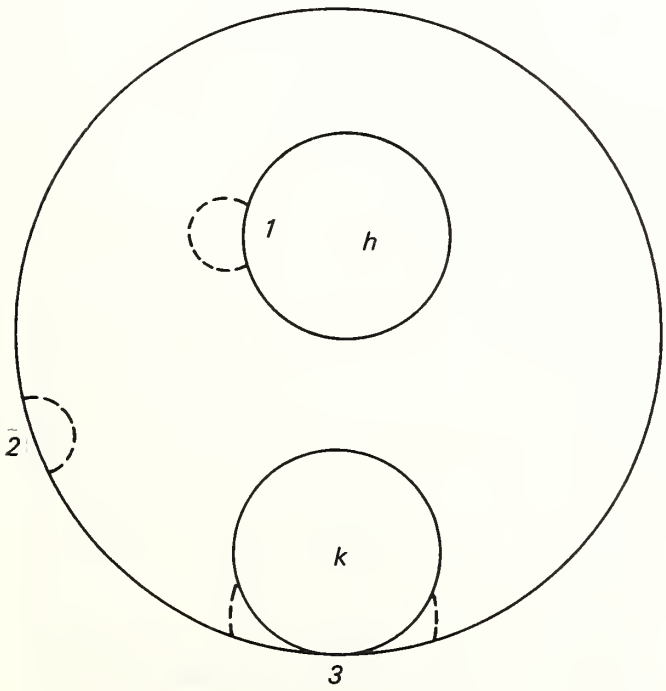
Figure I-29 illustrates some of the possibilities for singular neighborhoods on a disk. Each interior point of the disk has a regular neighborhood consisting of a small disk with center at the point. On the boundary, the neighborhoods consist of hemi-disks with the point on the diameter of the hemi-disk. At a singular point of the boundary, the neighborhood is more complex. It may be characterized in two ways: first, by the number of incident 1-cells at the singular 0-cell, and second, by the number of neighborhood components produced by removal of the 0-cell from the neighborhood.

The illustration shows a disk with two holes. The first hole is completely interior to the disk. At the point 1, of the hole h, there is a neighborhood that is a hemi-disk; similarly for point 2 on the boundary of the disk itself. On the other hand, the point 3 is singular. Removal of this point separates the neighborhood into two components; this is one-half the number of incident 1-cells, or one-half the degree of the vertex relative to the incident 1-cells.



Two examples of folding

Figure I-28



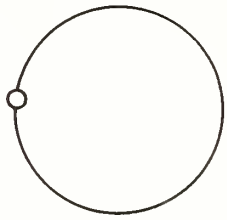
Neighborhoods on a disk

Figure I-29

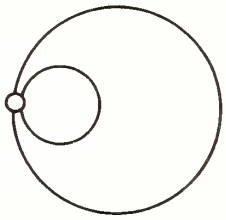
Representative Graph Codes

The following examples represent codes of simple graphs. A given graph code does not necessarily represent a unique graph; however, homeomorphic reduced graphs have the same codes. Each of the following examples represents a graph consisting of a single component, a situation sufficient for the purpose of explanation:

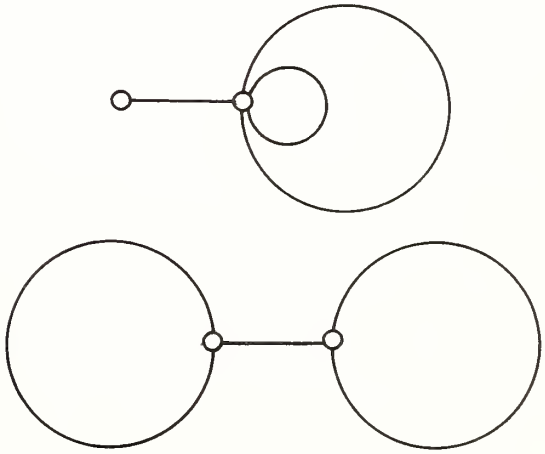
Code 1: (1, 1, 1, 0); A simple loop



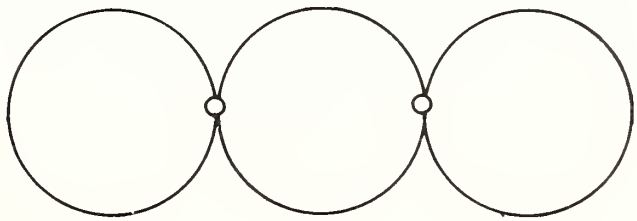
Code 1: (2, 1, 2, 0); A double loop



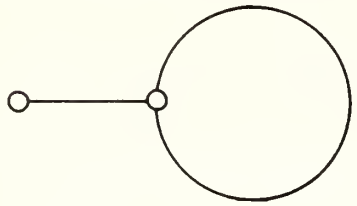
Code 1: (3, 2, 2, 1); A double loop with a tail, or two single loops joined by an acyclic 1-cell



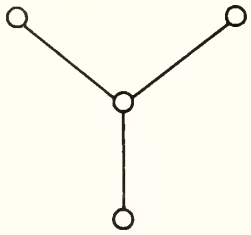
Code 1: (4, 2, 3, 0); Two loops joined by a 1-circuit



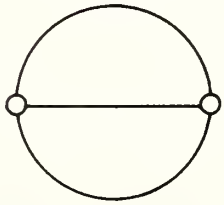
Code 1: (2, 2, 1, 1); Single loop with a tail



Code 1: (3, 4, 0, 3); Tree structure



Code 1: (3, 2, 2, 0); Two 1-circuits



Graph Analysis

Every editing procedure of ARITHMICON depends in some way on the analysis of a characteristic graph. From the discussion of neighborhoods, it can be seen that the internal structure of a 2-cell may be arbitrarily complex. For this reason, a completely general system of graphic analysis is necessary.

The analysis of a graph is simplified by the process of reduction. A graph is reduced by fusing each pair of 1-cells incident at a 0-cell of degree equal to two. The reduced graph of any pair of homeomorphic graphs has the same number of 1-cells and 0-cells.

The reduction of a graph is accomplished by a procedure called CHAINS. CHAINS is an algorithmic procedure operating on a fixed set of individual data sets. The resulting data set created is a description of the reduced graph.

CHAINS: Data set

The format for data set description is (index, record)

DIME: (segment, From-node, To-node, left block, right block)

Essential: (index, 0-cell, multiplicity)

Inessential: (index, 0-cell, multiplicity)

Chain: (pointer to segment)

Flag: (segment, flag = (0 or 1)

Chain Head: (index, chain number, pointer, initial vertex, . . . final vertex)

The pointer in Chain Head points to the first segment in Chain.

In functional notation, the first segment of a chain is

DIME (Chain (pointer (index_{CH}))); CH = Chain Head.

The algorithm has the structure shown schematically in figure I-30.

Initializing a Chain. Set the initial vertex equal to the first essential vertex having multiplicity $\neq 0$. If no such vertex can be found, exit from the procedure. Initialize the chain head record. Increment current chain number by 1. Set pointer equal to current value of chain stack pointer. Set initial vertex to vertex found in previous step.

Continuing the Chain. Find first segment for which From-node is end of chain. If none exists, find first segment for which negative segment satisfies the condition. Flag segment as chained. Push segment index to chain stack. Set current end of chain to To-node of last chained segment.

Testing for End of Chain. Is current end of chain an essential vertex? If so, set value into chain head record and push chain head record to chain head stack. Otherwise continue the chain.

Exit from the chaining procedure for chains on essential vertices either terminates the chaining procedure or else there remain chains of inessential vertices. These chains are necessarily circuits. Since the representation of a circuit requires at least one vertex, the procedure begins with the selection of an arbitrary vertex in the list of inessential vertices. The structure of the program is identical with that of the program for chaining on initial essential vertices, but with a different set of tests and procedures.

Initializing the Chain. Choose the first inessential vertex having nonzero multiplicity. If found, decrement multiplicity by 1, and set current end of chain to this vertex. If none such, the chaining procedure is ended. Set up Chain Read.

Continuing the Chain. Find the first segment for which From-node is current end of chain. If none such, then find first segment for which negative of segment satisfies the condition. Flag this segment as chained. Push index of segment to Chain stack. Set current end of chain to To-node of last chained segment.

Testing for End of Chain. Is current end of chain identical with initial vertex? If so, chain end has been reached. Set terminal vertex in chain head record to current end of chain. Push chain head record to chain head stack. Otherwise continue chain.

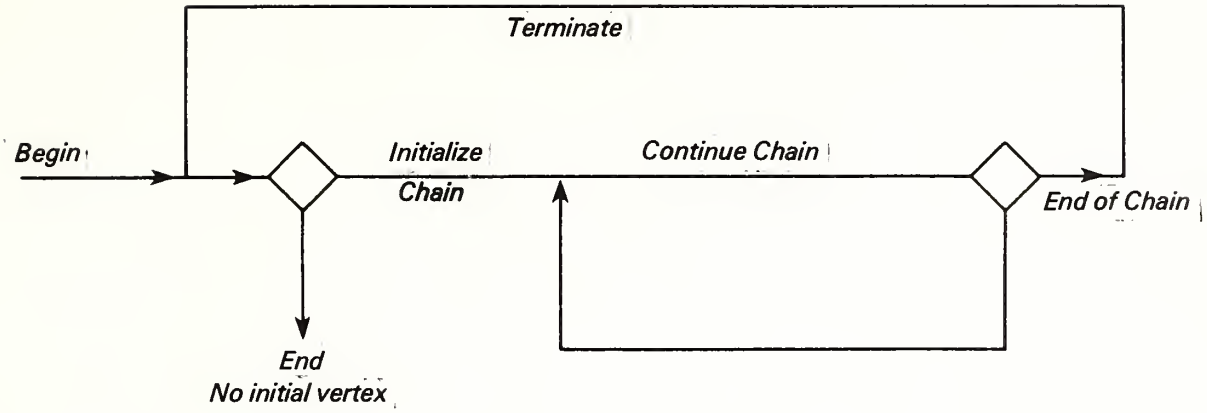
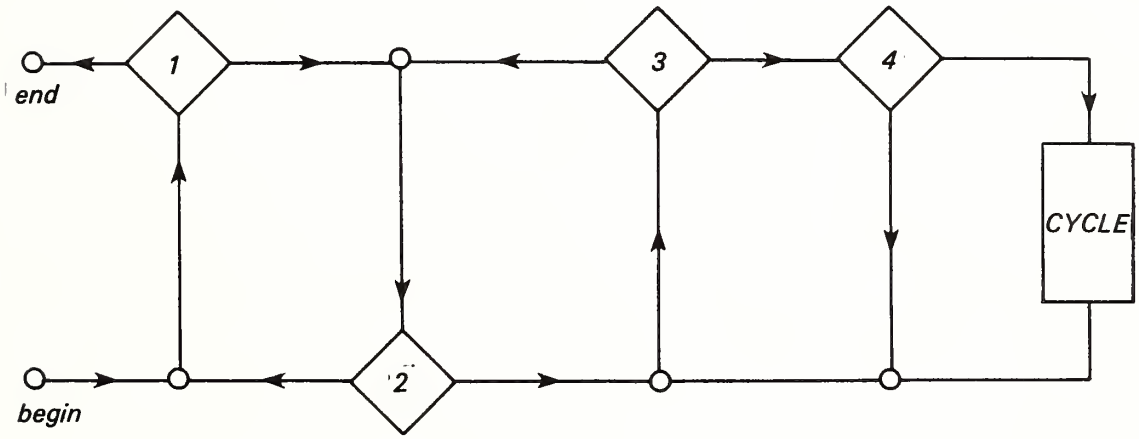
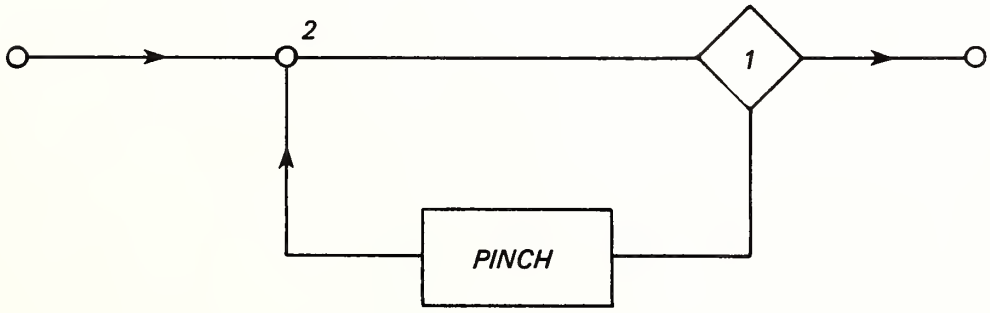


Figure I-30



KIRCHOFF



CYCLE

Figure I-31

The second phase of the analysis of a graph, after reduction, is a connectivity analysis. By simply counting 1-cells and 0-cells, the characteristic of the graph may be computed. From the characteristic, the number of 1-circuits may be determined directly. However, this simple procedure fails to identify those cells of a graph which belong to no cycle whatever. These are the so-called acyclic cells. The identification of these cells is carried out in a procedure called KIRCHOFF. The procedure, with only slight modifications, is due to G. Kirchoff, the originator of this type of graph analysis.

The object of the procedure is twofold: first, to identify the acyclic 1-cells of a graph, and second, to produce a graph code in the format specified for such a code:

$$\text{Component; } (n_1, n_0, \mu, \alpha)$$

where μ denotes the number of 1-circuits, and α, the number of acyclic 1-cells.

The description of the algorithm begins with a description of he data sets.

The input data is a list of segments related to their initial and terminal points. This is the list referred to as the output of the chaining program, except that the pointer to the chain stack is not used in the algorithm.

In the course of the analysis, the input record is extended by adding two fields: first, the component number in which the segment is placed, and second, a mark indicating a cyclic or acyclic segment.

In the course of the procedure, an independent count of the number of independent circuits is made. This count is compared with the number computed from the characteristic value as a check on the procedure.

Data sets for KIRCHOFF.
Segment: (From-node, To-node, Component, Cyclic)
Node list: (Node, pointer, M1, M2, segment)

In this last record, M1 and M2 are interpreted as the multiplicities of the associated node in a corresponding component.

The structure of the algorithm is shown in Figure I-31.

Component Initialization. Increment current component number (initially zero).

Select the first segment not assigned to any component (Comp. = 0). Mark the component field with the current component number. Select the From-node of the segment as the initial node. Place the node at index position 1 in the node list. Set the multiplicity, M1(1) = 1. Set the index pointers, i and j, each to the value 1. If no segment is unassigned, the procedure terminates.

Component Formation. Test 2 merely compares the pointers, i and j. If the value of i exceeds the value of j, control returns to the initialization procedure.

Test 3 is a search for a segment having the node, node (i), as a From-node. If none can be found the segments are searched after reversal. If no segment is found, the pointer i is incremented by 1 and control returned to test 2.

Test 4 is a test for the formation of a circuit by the last chained segment. The test consists of a search on an index, k, for a node (k) equal to the To-node of the last chained segment. For either outcome of the test, the multiplicity, M1(i), is incremented by 1. If no circuit is formed, the index, j, is incremented and node (j) is set equal to the To-node of the last chained segment. On the other hand, if the node, k, corresponding to the formation of a cycle has been found, the multiplicity, M1(k), is incremented by 1, and control passes to the routine CYCLE. Whenever a new node is added, that is, when no circuit is formed, the value of pointer (j) is set equal to the value of i. The value of the segment pointer is placed in the field, segment (j). Thus, each new node added to the node list is associated with its antecedent, via the pointer set equal to the index of the last chained segment. If a circuit had been formed, the index of the last chained segment would have been placed in the cycle-forming segment list.

The condition of the graph at the point where a circuit has been formed is shown in figure I-32.

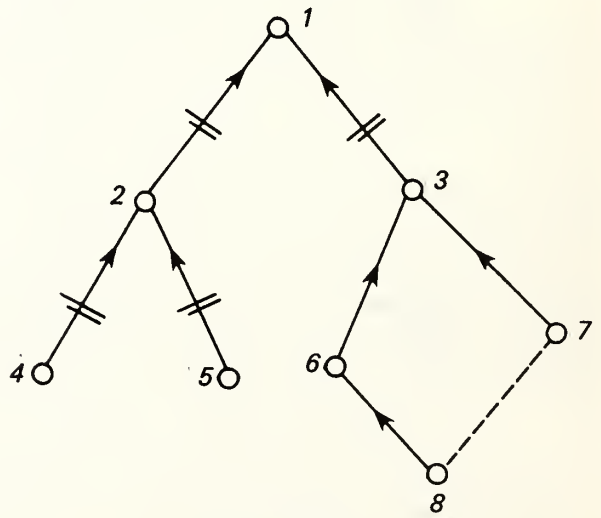


Figure I-32

The cycle-forming segment is shown as a dotted line. Note that each node except for the initial node has a single antecedent when the cycle-forming segment is removed from the graph. In the procedure CYCLE, to be described next, reference will be made to this figure. The free ends of the graph will be removed in a recursive procedure PINCH. What remains will be a point set connected by arcs into a simple loop. In figure I-32, the segments removed by the procedure PINCH are indicated by cut marks transverse to the segments to be removed. The segments remaining will be marked as cyclic in the cyclic field of the segment record. A segment once marked as cyclic retains this mark throughout the remainder of the procedure.

CYCLE. The structure of the procedure is shown in figure I-31. The condition of the graph at the beginning of the procedure is indicated in figure I-32.

The procedure begins by setting M2 = M1 for each vertex in the node list.

The loop for pinching off free ends is then entered. A search for a free end is conducted. For any M2 = 1, the value of M2 is decremented by 1. Now either the related node has an antecedent in the graph with M2 $\neq$ 0 or not. If so, the value of M2 (point (j)) is decremented by 1, j being the index of the free end. If the node (j) has no antecedent with nonzero M2, it has a successor satisfying this condition. This successor is found by searching the pointer list for the value j. When found, the corresponding M2 is decremented by 1. This pinching procedure is executed recursively until no free ends remain in the M2 list of multiplicities. At this point, the graph has been reduced to the set of uncut segments as shown in figure I-32.

An exit from the PINCH loop then transfers control to an exit procedure which lists the segments in the remaining 1-circuit of the graph. The first of these segments is the cycle-forming segment. The remaining segments are found by the following procedure. Beginning with the last node for which M2 $\neq$ 0, remove segments by first, removing and listing the cycle-forming segment. This reduces the value of M2 to 1 for two of the points. Then pinch off the remaining free ends, listing the segments as they are pinched. The process will terminate when the apex (the first vertex in order of precedence) is reached. At this point, this particular vertex will have M2 = 0. In fact, M2 will have been reduced to zero for every node in the node list.

The outcome of a graph analysis consisting of the procedures CHAINS, followed by KIRCHOFF, is a list of chained segments, the individual chains, the essential vertices, and the graph code. As used in the editing procedures, this analysis is accompanied by a graphic output wherever this may be required or selected.

An example of the output of a graph edit is shown in figure I-33. The format places the graph code in the upper left-hand corner of the image, directly under the identifier of the graph. The programs for annotation of a graph will be discussed at a later point. It is sufficient at this point to indicate that a graph may be selectively annotated. Each element of the graph may be identified by means of an interactive selection process. For the moment, the description of the general procedures of the edit will be continued.

First, each DIME segment must have a complete set of information fields. This assures one that each segment has two bounding 0-cells (possibly a duplicated 0-cell), and that each segment has two cobounding 2-cells (again possibly a single 2-cell duplicated). This means that one part of the requirement that the structure be a complex is satisfied, namely that each 0-cell is on the boundary of some 1-cell, and that each 1-cell is on the boundary of some 2-cell. The remaining part of the edit with regard to the conditions on a complex would require that each 2-cell has its boundary in the 1-skeleton. The format of the segment ensures that the 1-cells are all bounded in the 0-skeleton. The only editing procedure required would be the determination that each 2-cell was bounded in the 1-complex. This step is deferred to a later point in the edit in the interest of efficiency.

The editing procedure is an analysis of the fundamental neighborhood of each of the vertices on the map. This neighborhood is a collection of generalized 2-cells or disks. The dual graph of this vertex is first analyzed. If there are no singular segments, the coboundary of the vertex is a cocycle. In this case, the graph code of the dual graph is

1: (1, 1, 1, 0)

which signifies that there is a single component consisting of one 1-cell forming a single loop and a single 0-cell. The

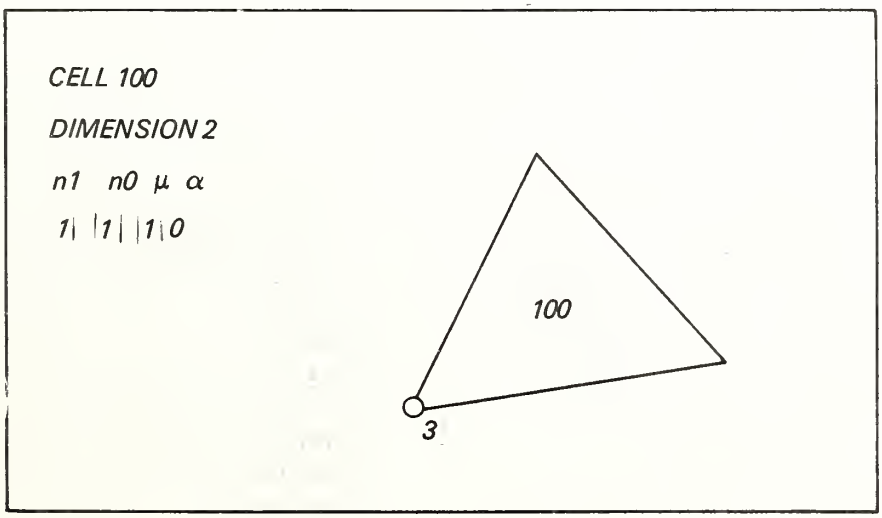


Figure I-33

presence of singularities will be signified by the presence of multiple loops as in the following code:

1; (n, 1, n, 0) .

In such a case, a graph will be drawn representing the pseudo-dual at the vertex.

The presence of acyclic elements, or the absence of cycles, always indicates defective code. In such cases, the graph is always drawn representing the pseudo dual.

In any case in which a defective graph appears or in which there is a code indicating a defective encoding, the corrections are made immediately. This part of the edit corresponds to a test of the condition that a two-dimensional manifold be topologically flat in the neighborhood of each vertex. This test is not a sufficient test. It is possible that some of the 2-cells abutting the vertex do not have complete boundaries in the closed fundamental neighborhood.

The next stage in the edit is to determine the boundary of the fundamental neighborhood. This boundary is the sum (mod 2) of the boundaries of the individual 2-cells. At this point in the edit, one has already established that the intersections of the boundaries of these cells are topologically consistent in their relation with their coboundary 2-cells; thus, to complete the boundary analysis, it is sufficient to analyze the boundary of the fundamental neighborhood.

The boundary of the fundamental neighborhood is a graph. It should be a set of circuits. In the simplest case (that which occurs most frequently) the boundary would be a simple loop. The presence of cyclic components indicates a disk with holes. The presence of multiple loops indicates that there are singular boundaries of 2-cells in the neighborhood. In such cases, the graph is drawn. This graph may be drawn at different levels of detail. First, one may draw the boundary itself (with or without annotation). Then, if required, one may request the detailed graph of all the graphic structure associated with the neighborhood. If desired, one may suppress embedded structure interior to individual 2-cells.

This analysis of the boundary of the fundamental neighborhood assures us that the neighborhood is a flat disk with possible holes, in short, a consistent geometric structure. It remains to determine whether or not the coordinate descriptions of the arcs are consistent with the topological description.

The first test is one that determines whether or not the vertex is interior to the region bounded by the boundary of its fundamental neighborhood. One of the most frequently occurring tracing errors places the point outside of this boundary. When an analytic indication of this condition is encountered, the editing program always presents a graph representing the vertex in relation to its boundary. ARITHMICON contains facilities for immediate correction of the location of the vertex.

The second kind of anomaly resulting from inconsistency between the topological and metrical descriptions is the presence of folds. Folds are detected in the analysis of the pseudo-graph of the coboundary of the vertex. Traversing the pseudo-graph in an arbitrary direction, any retrograde segment encountered will signify that a fold exists. Again, ARITHMICON contains the necessary facilities for correcting this condition. It should be emphasized that one must examine the pseudo-dual of the coboundary and not the boundary of the fundamental neighborhood, which may have any number of retrograde segments.

A neighborhood that satisfies a set of tests such as those listed above will be topologically and metrically consistent. Two such neighborhoods containing common cells will be consistent with regard to their intersections if each is consistent separately.

Map Editing

The purpose of map editing is the validation of the map model as a faithful representation of the source map. This is done by establishing a homeomorphism between the cells of the model and the cells of the map.

A Chain Mapping is a mapping between two cellular structures that is one-to-one, preserves dimension, and preserves incidence. The fact that incidence is preserved may be expressed in algebraic terms by the fact that such a mapping, denoted here by F, commutes with the boundary and coboundary operators. This commutativity is sometimes expressed graphically by a diagram such as that in figure I-34. In this diagram, the top row represents map cells, and the bottom row, the model cells. These are related by a one-to-one mapping as indicated by the double-headed arrows marked F.

To illustrate the significance of the diagram, consider a given 0-cell c^0. This cell is represented by the upper left point of the diagram. The coboundary $E_0^1(c^0)$ is represented by the first point to the right of the upper left corner. The image of this coboundary on the map is $F(E_0^1(c^0))$, which is represented by the point immediately below on the bottom row. But by the cummutative rule, this image is also given by the expression $E_0^1(F(c))$. This point may be reached by starting from c^0, moving to the image below on the model, and moving to the right by taking the coboundary.

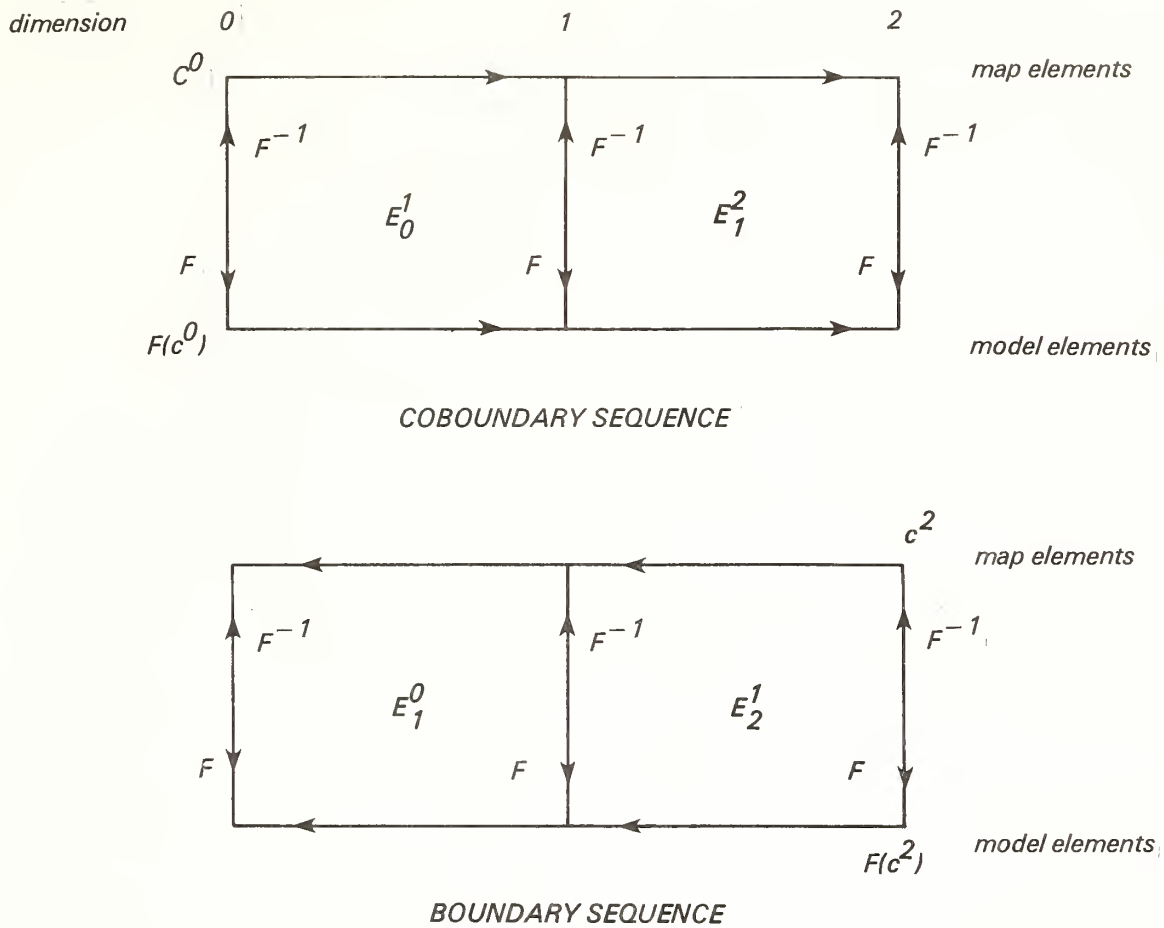


Figure I-34

The procedures used in conducting an edit may be traced on the diagrams of figure I-34. For example, to edit a boundary of a 2-cell, the dual of the sequence exhibited above is used. This sequence is traced on the boundary-diagram.

The general principles of map editing follow from this relational scheme. The edit of a fundamental neighborhood of a vertex is carried out in essence as a test of the commutatitity rule just explained. One compares the sets

$E_0^1(c^0)$; the map coboundary of the map-cell, c^0

with the set formed by the procedures

$F^{-1}(E_0^1(F(c^0)))$; the graphic image of the coboundary of the model cell corresponding to the map-cell.

These sets should be identical, since they are algebraically identical. If the model structure has a defective graph code, the sets cannot be identical and further investigation is made, primarily by direct comparison of the two images.

Precisely the same principle is followed when editing a 2-cell boundary. In this case, one is concerned with boundary rather than with coboundary operations. In either case, the graph code, when admissible, is taken as a sufficient characteristic of the cell neighborhood being tested. The graph code of the vertex may be immediately obtained by direct observation of the map. In the few situations in which the graph code is too complex for direct visual perception, the necessary graphic comparisons of the two images may be made.

In any case in which it is decided that the graph codes do not coincide, the next step is to obtain the symmetric difference between the sets being compared. This set is the 1-cells in one set but not in the other. These are 1-cells attached to the model 0-cell in error, and the 1-cells that should be, but are not, attached to the 0-cell.

The principal analytic tool is the graph analysis consisting of the sequential execution of the procedures CHAINS and KIRCHOFF. The summary of the analysis is presented as a graph code and an image of the graph itself. The first example is the analysis of a vertex coboundary, for which the code is

1; (1, 2, 0, 1)

the code for a nonsingular 1-cell with distinct boundary 0-cells. Tracing the course of this edit on the diagram of figure I-34, we begin with a 0-cell of the map, and through the index (representing the mapping F) locate the corresponding 0-cell in the model. The coboundary operator, E_0^1, is then used to retrieve the cobounding 1-cells from the model. It is this set of 1-cells that is analyzed to form the dual graph referred to above. Note that the retrieval of the DIME segments necessarily retrieves the associated incident 2-cells abutting the vertex. Thus corresponding to the original 0-cell on the map, the procedure retrieves the corresponding cells of all dimensions incident with the 0-cell.

The map is inspected in the neighborhood of the 0-cell. This amounts to looking visually into the neighborhood of the 0-cell, and hence the corresponding coboundary operation is implicit in the visual inspection. Suppose that the inspection reveals that there is a segment missing from the coboundary in the model. This segment may be identified by direct comparison of the graph from the model with the source map. From this segment on the source map one may identify the corresponding segment in the model by retrieving the coboundary of the opposite vertex. In the simplest case such a segment may be found, and will be correctable by inserting the correct vertex identifier.

Tracing the edit on the diagram, we see that we move to the right to examine the coboundaries of the vertex in the model and the corresponding vertex on the map. The error in the model is corrected by finding the correctable segment from information obtained from the coboundary on the map. This segment is isolated and corrected.

The second example shows a different use of the editing diagram. Let the vertex be analyzed and have graph code

1: (2, 2, 1, 1)

which represents a loop with a tail. In the simplest case, inspection of the map will reveal that there is an extra segment at the vertex in the model. One could simply detach the segment from the vertex by blanking the vertex code. It is more economical, however, to insert the correct code at this point. The correct vertex code is obtained by looking into the coboundary of the opposite vertex on the map. Then from the boundary of the appropriate segment we obtain the vertex code to be inserted in the corrected segment.

In each of the two cases just considered, the function F is employed to map the correct vertex identifier into the identified segment. It is instructive to trace out the sequence of events in detail. In the first case in which there is a missing segment, one in effect compares the coboundary $E_0^1(c^0)$ of the map with the image set in the model, $F(E_0^1(c^0)) = E_0^1(F(c^0))$. Since by definition of a chain mapping, the commutative rule holds between the chain mapping and the boundary and coboundary operators, the two versions of the coboundary should coincide. Since the model has a defective code, we know in advance that the rule cannot hold and that the symmetric difference of the two sets represents an error in coding. By adjudication of the differences, segment by segment, one identifies the miscoded segments by searching the relevant neighborhood.

This principle is applied generally when using the ARITHMICON system for model validation. One further example is important in practical applications. One has frequent occasions to validate one-dimensional structures and their fundamental neighborhoods, in connection with street-name edits. The chaining algorithm, followed by a graph analysis, will identify any components associated with a street name. At the end-points of components one may refer to the map to determine whether or not the termination is correct. Often it is not. In this case, the question arises as to how to identify the misnamed segments. To do this one retrieves the coboundary of the end-points of the components. Comparison of the associated street names with those on the map will serve to identify the misnamed segment. This procedure is of great practical value, since street-name variants, and spelling errors occur frequently in the coding of urban maps.

Representation of Chains and Operators

Chains will be represented as lists of pairs,

(coefficient, identifier)

each individual pair representing some cell of the chain. The coefficients will be signed integers. Addition of two chains will be carried out by sorting the two representing lists on identifier, and summing the coefficients of like terms. The addition, term by term, is always carried out as signed integer addition. The coefficient of a given cell within a chain may be positive, negative, or zero. The coefficients will be interpreted in three distinct ways. First, literally, as signed integers. Second, as integers modulo 2, in which case, the algebraic signs are ignored, and the integers reduced modulo 2, and finally in a sense of set membership, in which the coefficients are ignored. Each of these three interpretations has a practical use, and the interpretations lead to different results.

The lists which represent chains are furnished with pointers to the top of the list. Erasure is carried out by resetting the list pointer to zero. The flow of data in the retrieval system is shown in schematic form in figure I-35. The system consists of a set of lists, representing chains, and a set of operations, representing the boundary and coboundary operators. Lists and operations are related as indicated in the flow diagram. This diagram is a graph, with nodes representing lists, and segments representing operations.

The lists, cellist, seglist, and inlist, represent chains. Cellist represents either a 0-chain or a 2-chain as determined by a system parameter, dimension. Seglist and inlist always represent 1-chains. Toplist, when associated with inlist, forms a DIME representation of a 1-cell.

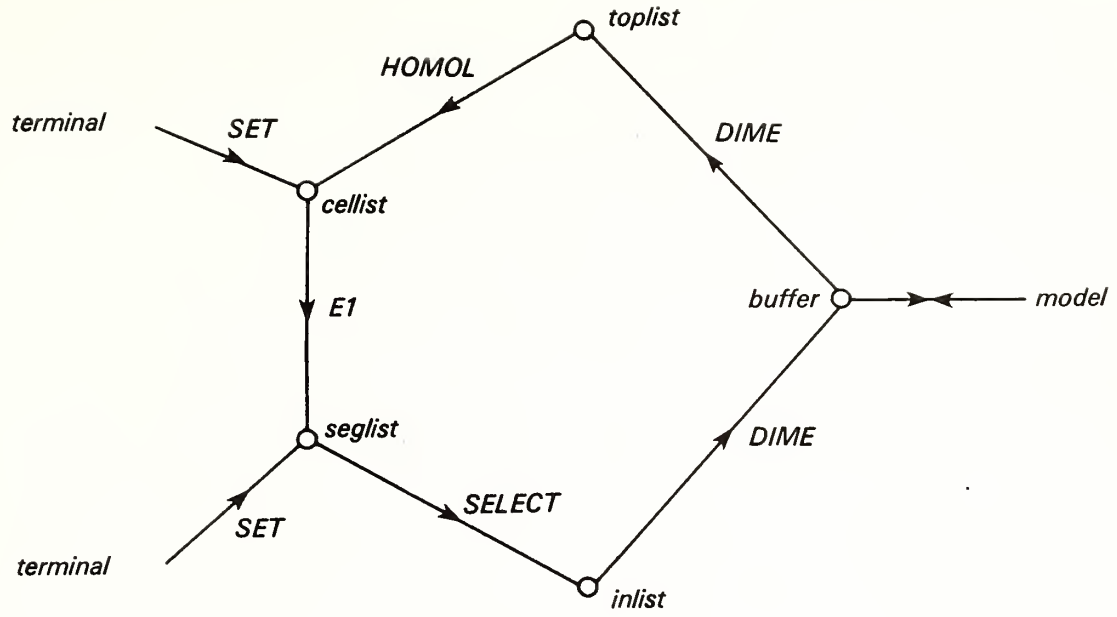


Figure I-35

The system operations which control the flow of data through this sequence of lists are defined in the following list. The definitions are in terms of already defined operations.

$$E1(d, list1, list2) \equiv list\ 2 \leftarrow list2 + E^1_d(list1);\ d = 0, \text{ or } 2.$$
$$DIME\ (list1, list2) \equiv list2 \leftarrow (E^0_1(list1), E^2_1(list1))$$

These three functions complete the implementation of the boundary and coboundary operations. Note that inlist and toplist, taken together at the end of a retrieval cycle, always represent a list of DIME segments. In addition to these fundamental operations, there are three auxilliary functions:

$$SELECT\ (PROC, list1, list2) \equiv list2 \leftarrow PROC\ (list1)$$
where PROC is some list processing procedure.
$$ERASE\ (list) \equiv list \leftarrow null$$
$$SET\ (list1, list2) \equiv list2 \leftarrow list1$$

The structure of the retrieval system as shown in figure I-35, may also be represented as an event diagram, figure I-36. Any path starting at the initial node of this diagram and terminating at the final node represents a data retrieval sequence. The retrieval loop is closed through the function

HOMOL (d, toplist, cellist, field); this function transfers the list of cells of the designated dimension and field from toplist to cellist.

Example 1. Construction of a fundamental neighborhood of a vertex.

ERASE (ALL)
SET (dimension, 0)
SET (cellist, c^0)
E1 (d; cellist, seglist)
SELECT (ALL, seglist, inlist)
DIME (inlist, toplist)

At this point of the sequence, the DIME segments of the coboundary of the 0-cell c^0 are in toplist. The analytical programs are then executed:

CHAINS (toplist, chain-head)
KIRCHOFF (chain-head)

These programs provide a graph analysis of the 1-complex represented in toplist. Resuming the retrieval sequence at the end of this analysis,

ERASE (cellist, seglist, inlist)
SET (d, 2)
HOMOL (2, toplist, cellist)

At this point, a cycle of retrieval is completed. The 2-cells abutting the 0-cell c^0 are now in cellist:

E1 (d, cellist, seglist)
SELECT (ALL, seglist, inlist)
DIME (inlist, toplist)

At this point, the boundary DIME segments of the 2-chain in cellist are in toplist. This graph may now be analyzed in the analytical programs. Continuing with the retrieval example:

ERASE (inlist)
SELECT (coefficient $\neq$ 0, seglist, inlist)
DIME (inlist, toplist)

At this point the boundary segments of the fundamental neighborhood are in toplist. The sequence

ERASE (inlist)
SELECT (coefficient = 0, seglist, inlist)
DIME (inlist, toplist)

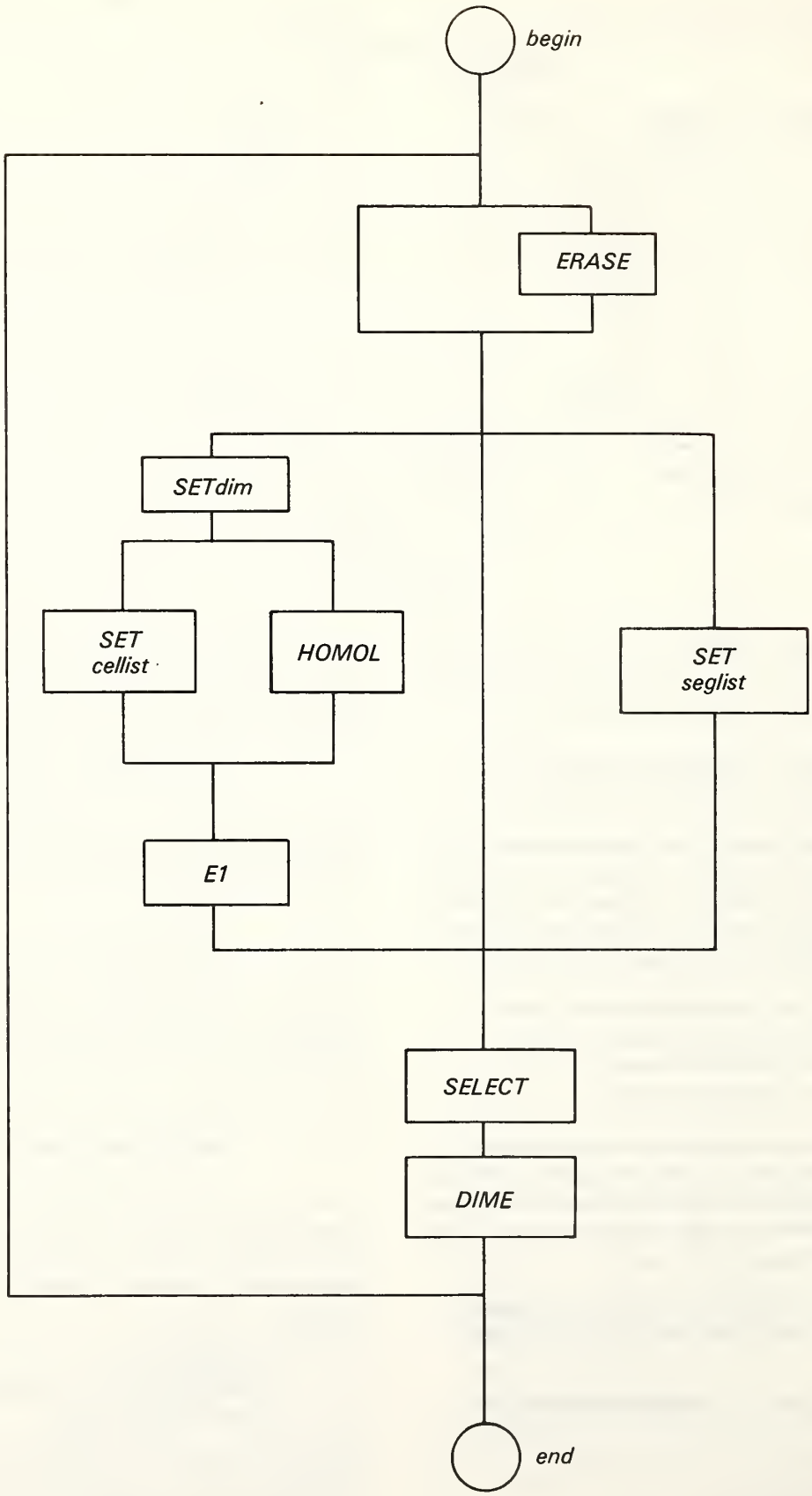

begin
ERASE
SETdim
SET
cellist
HOMOL
SET
seglist
E1
SELECT
DIME
end

Figure I-36

will place the internal (embedded) structure of the fundamental neighborhood in toplist.

In part II, in the section on the SPEAK language, it will be shown how long strings of commands of this kind are set up as permanent job-streams. For the moment, in the interest of clarity of explanation, the full sequences are displayed in detail. The successive stages of a retrieval of a fundamental neighborhood are shown in figures I-14, I-15, and I-16.

Example 2. Listing the 2-cells interior to a 1-circuit.

. This is an important data retrieval sequence in practice.

ERASE (ALL)
SET (seglist, 1-circuit)
SELECT (ALL, seglist, inlist)
DIME (inlist, toplist)
HOMOL (2, toplist, cellist, Left)

This sequence begins with the oriented 1-chain in seglist. It is transferred to inlist, and the DIME segments retrieved. The HOMOL function transfers the left-hand 2-cells to cellist.

SET (d, 2)
E1 (d, cellist, seglist)

Seglist already contains the original boundary 1-circuit. This is added to the boundary chains of the set of 2-cells in cellist. The result (mod 2) is the homologous 1-circuit, which together with the original 1-circuit now forms the boundary of a (possibly singular) annular ring. The newly derived homologous 1-circuit has the opposite sense to the original 1-circuit. Its sense is therefore reversed by multiplication of the cells of the list by -1.

Iteration of this procedure will result in the reduction of the list of 2-cells in cellist to a single 2-cell, which terminates the procedure, and the list of contents, or else the list will begin to repeat 2-cells in the manifold.

Figure I-37 illustrates the first stage in the formation of the homologous 1-circuit.

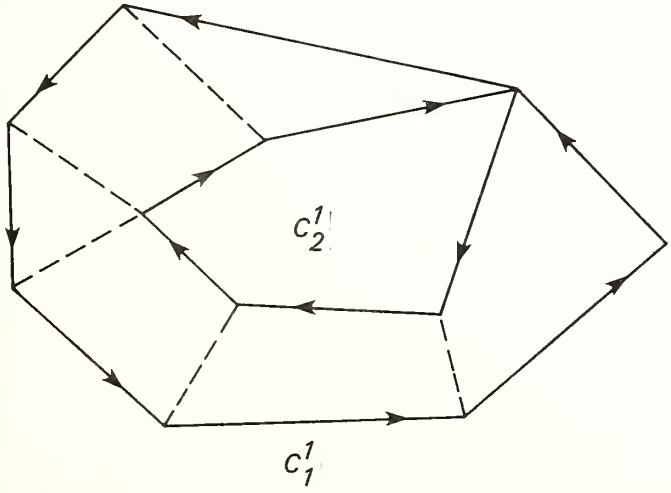


Figure I-37

In this figure, the original boundary C_1^1 and the homologous boundary C_2^1 are shown together with the (singular) annular ring of 2-cells which they bound.

Example 3. Construction of the adjacency matrix, A.

The adjacency matrix is the square matrix forms from the lists of adjacent vertices of a graph. Each row of the matrix is a list of vertices adjacent to a particular node of the graph. This matrix is related to the incidence matrix in the following way. If E is an incidence matrix, and E' its transpose, and I the identity matrix, then

$$A = E\,E' - I$$

The construction of this matrix in ARITHMICON is given in the following retrieval sequence. The related diagram is shown in figure I-38.

ERASE (ALL)
SET (d, 0)
SET (cellist, c^0)
E1 (d, cellist, seglist)
SELECT (ALL, seglist, inlist)
DIME (inlist, toplist)
ERASE (cellist)
HOMOL (0, toplist, cellist, To)

At the end of this cycle, the adjacent vertices relative to the original 0-cell are in the cellist. The connecting segments are in both seglist and inlist.

E1 (d, cellist, seglist)

At the end of this procedure the linking segments have coefficients equal to zero mod 2.

ERASE (inlist)
SELECT (coefficients $\neq 0$, seglist, inlist)

This places the linking segments not incident with c^0 in inlist.

DIME (inlist, toplist)
ERASE (cellist)
HOMOL (d, toplist, cellist, To

This sequence completes the second cycle. The cellist now contains the list of 0-cells incident at the second removal from the original cell. The associated coefficients represent the number of distinct paths from the original cell to the corresponding cell in cellist.

The iteration of this procedure will produce the powers of the adjacency matrix, row by row.

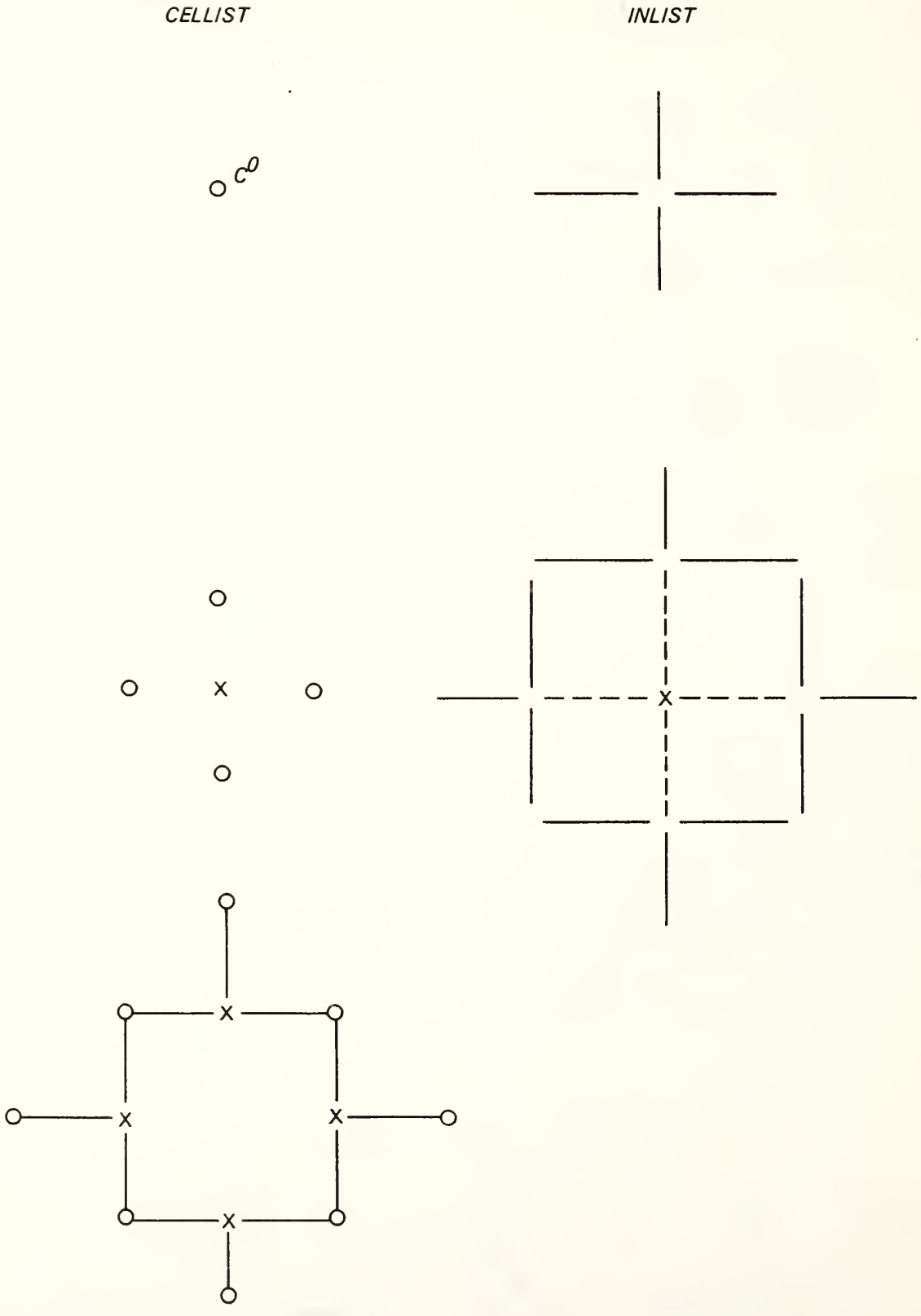

CELLIST
INLIST
C^0
Erased 0-cells indicated by x
1-cell coefficients = 0 mod 2
shown as dashed lines

Figure I-38

Example 4. The fundamental neighborhood of a 2-cell.

ERASE (ALL)
SET (d, 2)
SET (cellist, c^2)
E1 (d, cellist, seglist)
SELECT (coefficient $\neq$ 0, seglist, inlist)
DIME (inlist, toplist)

This sequence places the DIME segments corresponding to the boundary 1-cells in toplist. The sequence that follows collects the embedded 1-cells:

ERASE (inlist)
SELECT (coefficient = 0, seglist, inlist)
DIME (inlist, toplist)

Example 5. Identification of a 1-cell from its terminal vertices.

ERASE (ALL)
SET (d, 0)
SET (cellist, (c_1^0, c_2^0))
E1 (d, cellist, seglist)
SELECT (coefficient = 0, seglist, inlist)

At this point, the required 1-cell identifier is in inlist.

DIME (inlist, toplist)

This completes the DIME segment being sought. At this point the equality of the 0-cells (bounding 0-circuit) in toplist and in cellist may be verified.

Part II. ARITHMICON: A System of Computerized Cartography

SYSTEM OBJECTIVES

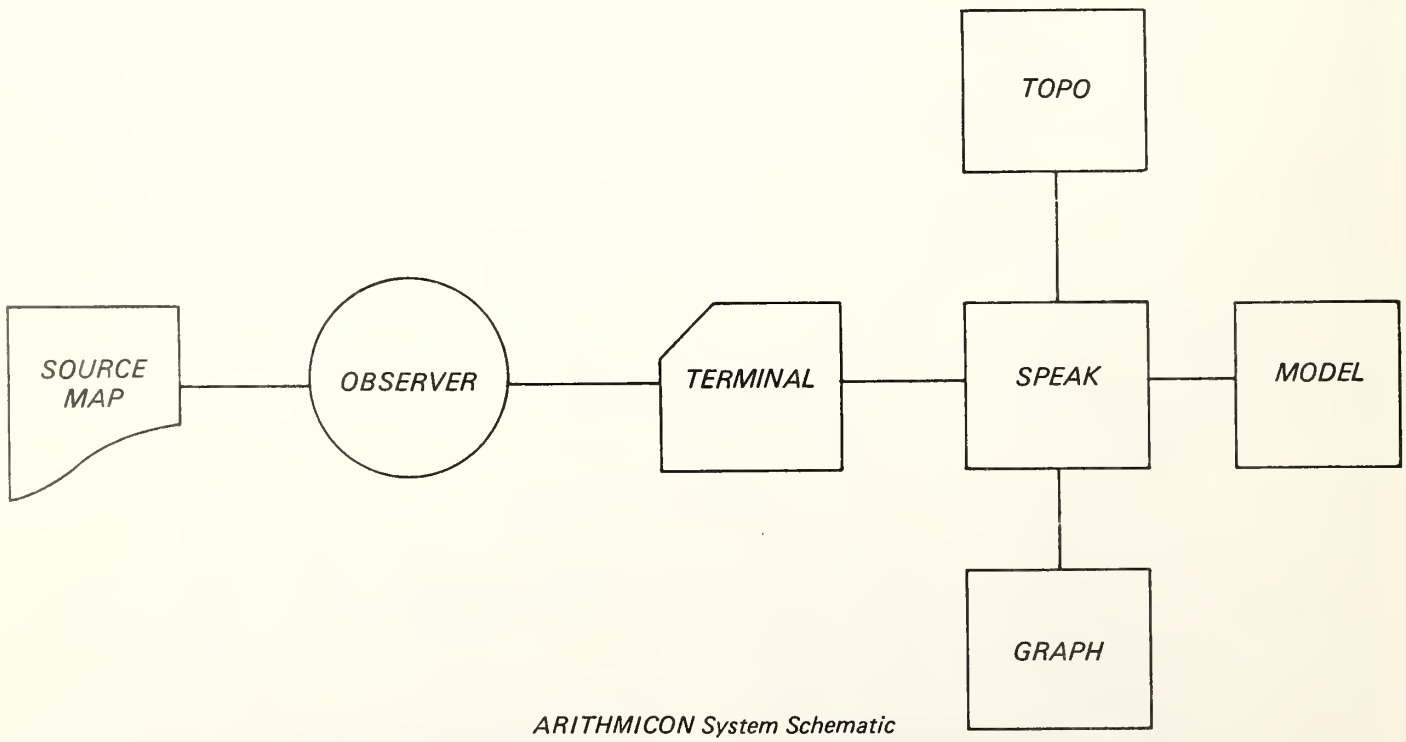
The primary objective of the system ARITHMICON is to provide a complete system of support for the topological analysis described in Part I of this monograph. To do this one must provide the means by which an observer may establish a hemeomorphism between a source map and an arithmetical model of such a map. The necessary procedures for this purpose have already been described in geometrical and algebraic terms in Part I. The means by which these procedures are to be realized as an actual computerized system are the subject of this discussion.

A secondary objective, but an extremely important one, is to provide a system that has the potential for wide distribution. The achievement of such an objective makes it possible to decentralize the work of constructing and editing computerized cartographic models. In line with this objective, the system has been designed in such a way that it may be implemented in most general microcomputer configurations, and with only moderate requirements for memory. One indispensable requirement is for the use of a terminal capable of supporting interactive graphic procedures.

The original realization of the system in terms of operating software was written in MACRO-10, and subject to roughly 2 years of actual operating experience. Although this version resides in a DEC-10, a relatively large and powerful general-purpose computer, it is of moderate size. Any version based on the documentation of the present monograph should realize important economies in the use of random access memory. Some form of disk memory is required, but for most applications, even a floppy disk would be sufficient.

SYSTEM COMPONENTS AND FUNCTIONS

The principal system components and their mutual relations with one another are indicated in the schematic of figure II-1. The elements labeled SPEAK, TOPO, and GRAPH are the subsystems of ARITHMICON. The model is, of course, the model described in part I. The terminal must be capable of both iconographic and iconoscopic functions; that is, it must not only produce images, but it must accept graphic inputs as well. The graphic terminal is the interface between the observer and the model.



ARITHMICON System Schematic
Figure II-1

In operation, the observer must compare the transmitted images at the terminal with the actual image of the source map. The observer may alter and annotate the source map, and may also alter and annotate images on the terminal. Such images and annotations are transmitted to ARITHMICON for interpretation and analysis.

The system TOPO is essentially a computerized representation of the procedures of algebraic topology as applied to combinatorial manifolds.

The subsystem GRAPH provides the encoding and decoding of all graphic communication between the observer and the system.

The system is under the control of the subsystem SPEAK. This subsystem coordinates the flow of data between the terminal and the model, and directs the analytic subsystem TOPO and the graphic subsystem GRAPH.

THE SUBSYSTEM SPEAK

The subsystem SPEAK and the language SPEAK are denoted by the same name. This will lead to no confusion, however, since it is always obvious from the context which sense of the word is meant.

The subsystem SPEAK is a system of procedures which supports the computer implementation of the language. It will result in greater clarity if the language SPEAK is described at the outset. This will make clear the reasons for the various functions of the several modules of the support subsystem.

The SPEAK Language

The SPEAK language below is modeled after the description of a language given by Paul C. Rosenbloom, **Elements of Mathematical Logic**, Dover, 1950. However, the same essentials are to be found in many of the standard texts on the subject of mathematical logic.

A language consists of a set of signs called an alphabet. By the term string is meant a finite sequence of these signs exhibited by writing signs in linear order from left to right. Strings will be named by Greek letters, α, β, and λ. If α and β are strings, then $\alpha\beta$ will denote the string consisting of the signs of the string β written in order following the signs of the string α. Two strings are accounted as the same if they have the same signs in identical places and the same number of places. A string formed from a given string by choosing the first k-symbols in order is called a head, and a string formed by choosing the last k-symbols in order is called a tail.

In this formulation, the signs of the alphabet will be called words. Words are classified as function words, or as constants or variables. The function words will be denoted by capital letters, F, G, . . . , the constants by letters, a, b, c, . . . , and the variables words by letters, x, y, z, ... A function word will be associated with a degree, a nonnegative integer.

TERMS. A term is a string formed in such a way as to satisfy a particular set of rules.

1. If F is a function word of degree n, and the words $w_1, \ldots, w_n$ are terms, then

$$F(w_1, \ldots, w_n)$$

is a term. The convention of enclosing the argument word list in partentheses will be retained, even though the construction of terms makes this unnecessary. The use of parentheses will make more complex terms more easily readable.

2. Any variable or constant word is a term.

RANK. A signed integer, known as the rank of a word, is associated with each word. For function words, the rank is one less than the degree. For constant or variable words, the rank is defined to be −1. The definition of a term implies that the rank of a term is equal to −1. The rank of any string is defined to be the sum of the individual ranks of the individual terms.

Terms may be characterized by the following two rules:

1. The rank of a term is −1.
2. The rank of any head of a term not equal to the entire term is nonnegative.

An alternative characterization of a term is the following: A string is a term if and only if its rank is equal to −1 and the rank of every tail is negative.

In the SPEAK language, the only constructions used are terms or abbreviations for terms. Particular abbreviations will be introduced into the discussion at appropriate points. All of the data retrieval procedures described in section 5 of part I are simple terms. Thus,

ERASE (inlist)
EL (cellist, seglist)
SELECT (ALL(Seglist), inlist)

each represents a term in SPEAK.

INTERNAL WORD FORMAT. The interal format of a word consists of two data fields:

word: (rank, identifier)

The language processor obtains the information for the rank either implicitly as when there are numerical inputs or explicitly by consulting tables of symbols.

INDIRECT REFERENCES. A means is required for distinguishing between a reference to an integer, such as 5, and the index or address, 5. This may be done by

introducing a function for indicating indirect reference, ∫(word). For visual convenience, this will be written informally as

word'

Thus, the word 5' refers to the contents of the index position 5. Some computers provide for a single stage of indirect reference; others allow for an indefinite number of iterations of the operation.

The language processor will interpret the symbol followed by the prime indication as an abbreivation for the use of the "∫" function. The prime will be written as a suffix even where left-hand Polish notation is used, except for situations in which this might lead to confusion.

The convention is particularly convenient for describing stack operations. A pointer, p, refers to an address of a pointer. Thus, p' refers to the actual pointer data, held at the address p. The reference p'' refers to the data at the top of the stack pointed to by the pointer p'.

A language such as SPEAK is known as a simple language because of the simplicity of its rule structure. Such a language may be parsed in a pushdown stack. The stack operations are standard, PUSH and POP. A combination POPSH, which transfers data by popping from one stack and pushing to another, is also used for abbreviation. The POP operation is designated by the upward-directed arrow; the PUSH, by a downward-directed arrow. A left-directed arrow denotes the operation of moving data designated in the second argument to the memory position designated by the first argument.

↓ (p, data) ≡
p ← p' + 1
p'' ← data

The stack pointer, p, is first incremented by one, and the data word is transferred to the top of the stack.

↑ (p, data) ≡
data ← p''
p ← p' − 1

The word at the top of the stack is transferred to the location designated by "data", and the stack pointer is decremented by one.

A useful abbreviation combines the POP and PUSH operations, popping from one stack and pushing to the other.

↑↓ (p, q)

Parsing a word string verifies that the string represents a valid term, or else the parser returns an error indication at the first point at which a syntax error is detected.

In parsing, terms are formed at the top of the n-stack. The function word and its arguments which form the term constitute a string of rank −1. The executable function may be regarded as a replacement operation, the words making up the string being replaced by the functional value, itself a word of rank −1.

In certain cases, these rules are apparently violated, as when terms are removed from one output stack to an auxiliary stack. In reality the rule is strictly adhered to. The auxiliary stacks may be regarded as parts of the output stack.

The parser will accept or read words from right to left, since the notation is left-hand Polish. Appendix A of part II contains a summary of the syntactic rules for formation of SPEAK procedures.

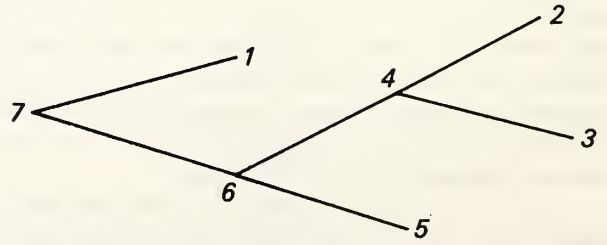
The parsing program accepts strings of input words. As previously explained, each word has two fields, an identifier field, and a rank field. The function RANK extracts the rank field from a word.

The Pushdown Parser

```
PARSE ≡
        ↓ (m, p')   ; store current p-pointer
        ↑ (m,  ), ← (p, m'') ; set new pointer
        WHILE (HEIGHT (p' ≠ 0)
          ↑↓ (p, n)
            IF (RANK (n'') ≠ −1 )
            THEN (JSR (n'') )
            ENDIT
          ENDWHILE
          ↑ (m, p)
```

Note: Inessential parts are omitted, for example, error returns from detected syntax errors. The PARSE is always executed from the n-stack. The pointer is popped from the n-stack to the p-pointer after memorizing the p-pointer. At the end of the parse, when stack height is zero, the original pointer is retrieved from the m-stack.

The PARSE procedure is reentrant. The structure of a system of stacks is then illustrated by the tree structure below. Beginning with a single stack, each stack may contain an ordered string of stacks. Such a structure to three generations of nesting is shown in the following illustration:



Any parse is executed at the point at which the command

JSR (PARSE term)

is executed. Upon return from this subroutine, the control returns to the parsing routine with the pointer reset to its value at the time of the call. When all unfinished parsing business from the m-stack has been completed, and when the original command stack is empty, the system returns control to the monitor, which requests additional command data.

The numerical indications on the preceding diagram refer to the order of completion of the parsing of the individual stacks.

Note: In the version of PARSE presented above, there is no mention of the explicit address of the argument. In use in SPEAK, a parse will always take its argument from the top of the n-stack. Upon entering the procedure, the argument will always be popped from the top of the stack.

The SPEAK Stack System

The system stacks used by SPEAK are shown in figure II-1. Any of these stacks may grow without bound. When a stack exceeds the allowable height, its lower half is stored in a B-tree. The root data together with the call to retrieve is left at the base of the stack, and the upper half of the stack is block transferred to the base. This procedure is done automatically, and the user need not be concerned.

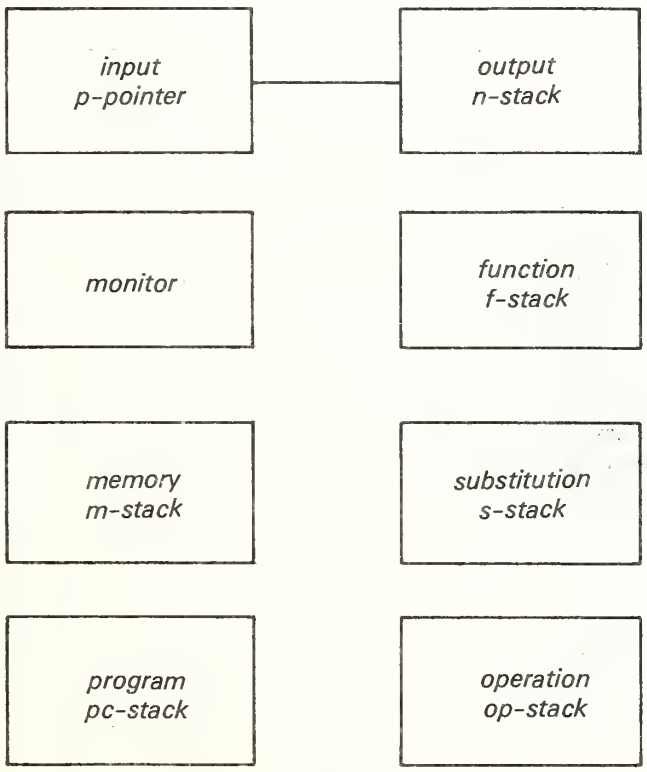


Figure II-2

Any number of stacks may be held in the general memory. The stacks indicated in the schematic are the active stacks of the system. The input stack is the current stack from which strings are being parsed. The monitor stack is the SPEAK monitor. The memory stack holds the addresses of stacks at which parsing interrupts have been generated (unfinished parsing business). The program stack holds the pc-addresses for returns from subroutines.

The function stack (f-stack) is not absolutely necessary but it is a convenience. It holds the governing values of the results of evaluation of successive decision functions.

The substitution stack holds the substitute values to which the corresponding variables are bound.

The op-stack holds stack addresses of the input stack for the purpose of memorizing reflexive references, and it holds stack addresses of the s-stack for memorizing the number of bound variables.

The monitor stack is the monitor program. Parsing this stack is the method of executing the monitor routine.

PARSING. The argument for the procedure PARSE will always be the top of the n-stack. The argument "stack" appearing in the calling sequence of PARSE will be n'', the effective call being

PARSE (n'')

Stack pointers will always be set by transferring the value of the pointer from the base of the stack, at which the pointer, consisting of two words, will be stored:

base: (height, topstack)

where topstack = base + height.

Thus the p-pointer will always be set by the transfer

↑ (n,), ← (n'', p)

The first system stack to be created is the command stack. This stack is the output of the language processor. The system monitor always begins execution of a command string by processing the term

PARSE command.

The parsing program places the word "command" in the n-stack, and the parse is begun in the manner indicated above. The words encountered in the sequel have all been accounted for in the previous explanations. A word at the top of the input stack is either a numerical argument, an executable function, or a word string required to be parsed.

IF THEN ELSE ENDIF. The SPEAK construction based on these four functions is

ENDIF, ELSE ⟨term3⟩, THEN ⟨term2⟩, IF (term1)

The terms in angle brackets are single-word labels for strings.

These functions will be described in the SPEAK formalism. Where functions used are intended as equivalent functions in NATIVE, the equivalents will be enclosed in corners, "⌜ ⌝".

The IF function transfers the top of the n-stack to the f-stack:

IF ≡ ↑↓ (n, f)
RETURN

The conditional execution, THEN, is as follows:

THEN ≡
⌜IF⌝(f'' ≠ 0)
⌜THEN⌝(PARSE (n''))
⌜ELSE⌝(↑ (n,)
⌜ENDIF⌝

Note: the functions in corners are NATIVE versions of the usual operations of structured programming.

The ELSE function is identical in structure with the THEN function differing only in the substitution of 1 for 0 in the conditional operation.

ENDIF ≡ ↑ (f,)

CONDITIONAL PROCEDURES. The arguments of the functions THEN and ELSE are procedures to be executed subject to the governing decision function. The outcome of the governing condition is at the top of the f-stack.

The signs for such a conditional procedure are enclosed in angle brackets. Thus

⟨ procedure ⟩

where procedure is a string. The first encounter, with the right-angle bracket, initiates a text handling procedure. This procedure forms a word consisting of the two fields of a pointer. The top of the stack is the word immediately to the right of the right-angle bracket, and the bottom of the stack is the final-angle bracket. This stack pointer is placed on the top of the n-stack.

The net effect of this is to provide a reflexive reference. Execution of an instruction,

THEN

will, if the governing conditional is satisfied, execute a parse of the stack designated by the pointer at the top of the n-stack. This parse will terminate at the base of the stack, corresponding to the left-angle bracket, and return to the main sequence of the parse.

REFLEXIVE REFERENCES. The bracket symbol,], will denote a reflexive reference to the input stack. At the time at which the rank of the symbol is recognized, the p-pointer points to the word immediately to the left of the bracket.

] ≡ ↓ (p', op)

The function DO is defined as

DO ≡
⌜IF⌝(f'' ≠ 0)
⌜THEN⌝ (PARSE (n''), ↑↓ (op, n), ↑ (f,))
⌜ENDIF⌝

The function, [, is defined as

[≡
↓ (op,)

Note: the DO operation always implies an IF operation within the scope of the reflexive operation. This condition is known as the stopping rule. The stopping rule is always executed before the DO operation is reached. Thus,

[DO, THEN ⟨ stack2 ⟩, IF (stack 1)]

is equivalent to the DO-WHILE construction of structured programming.

A second example is the string

[DO, IF (stack2), stack1]

This operation first results in a parse of stack 1, and then executes an iteration if the outcome of the decision function parse is set (= 0). This function is equivalent to the DO UNTIL operation in the convention of structured programming.

SUBSTITUTIONS. The indication of a substitution string is the use of brackets,

{ string1 ; string2; . . . . ; stringn }

It is assumed that each of the strings named represents a term. The semicolon designates a function defined as,

; ≡
↑↓ (n, s)

The right-hand bracket, "}" is defined as

} ≡
↓ (op, s')

The left-hand bracket, "{", is defined as

{ ≡
↑ (op, s')

Parsing a string of this kind, begins with the encounter with the right bracket which memorizes the position of the s-pointer. The parse continues until the first occurrence of ";", at which time the value at the top of the n-stack is popshed to the s-stack. This procedure continues as long as there are semicolons in the string. The final string is assumed to contain the "free" variables, x1, x2, . . . , xn. Any x-reference is an indirect reference to the substitute stack, s-stack, counted from the base of the stack.

As an example of a simple substitution, the following string is presented:

$$\{ + (x1, x2); 3; 4 \}$$

Parsing this string causes the following sequence of events. The encounter with the right-hand bracket causes the s-pointer value to be memorized in the op-stack. Then, in turn, 4 and 3 are first transferred to the n-stack and from there to the s-stack by the action of the semicolon operation. The s-stack therefore contains the numbers 4 in the first position and 3 in the second. Continuing the parse, one encounters the variable x2. This has been interpreted by the language processor as

$$2\ (n1)$$

which replaces x2 in the n-stack by the value n2. A similar evaluation of x1 is made in its turn. The ultimate answer which appears on the n-stack is the integer 7.

At the termination of the substitution, the left bracket resets the s-stack pointer to its original value.

The writer can obtain the first undefined x-value by means of a system interrupt, and so there is no need to remember the number of currently defined variables.

FREE AND BOUND VARIABLES. A variable symbol within the scope of a substitution operation is bound by this operation. A "free" variable is a variable for which no numerical value has been assigned. Any numercial value, however defined, as a simple number, a number theoretic function, or as a function of functions, may be specified as the assigned value of a variable.

Variables are denoted by the symbols x1, x2, . . . , xn, . . . up to any required finite number of such symbols. These, of course, may be denoted by any other letters thought more convenient, by the process of symbol equivalence. Free variables in a term are often encountered in procedures for which arguments are unspecified. Such procedures are called formulae. They become executable procedures when the free variables are bound within the scope of some substitution operator.

In an expression of the sort

$$\{ f(x4, x5, x2); term1; term2 \}$$

the variable x2 is free. The variables x4 and x5 are bound by the substitution. Their values are obtained by evaluating the terms term1 and term2, respectively. Before such an expression may be evaluated, it must be part of some larger context in which the variable x2 is bound by a substitution operation. An attempt to refer to a free variable during a parsing operation will result in an error return, since the reference will refer to a value outside of the range of the s-stack. When the language processor encounters an expression of the kind exhibited above, it is verified that the variables x1 through x3 are already defined. Otherwise, an error return is made.

ITERATION OF A SUBSTITUTION. If in the substitution operation

$$[DO\ (op), THEN\ (stack2), IF\ (stack1)]$$

the decision function is

$$n''$$

and the THEN component is

$$stack2 = \{ -(x1, 1), f(x2)\ ;\ ; \}$$

and the initial values on the n-stack are

$$(n1, n2) = (m, a)$$

then, the function computed by the construction is $f^m(a)$.

Since n'' is the decision function, the initial value is m ($\neq 0$). The decision value d = 1 then appears at the top of the n-stack. The IF function removes this value and places it at the top of the f-stack. The label of stack2 then is placed on the f-stack. The THEN function causes a parse of the stack2. This results in the computation of f(a) and the decrementing of n'' by 1. The values at the top of the n-stack are then

$$m - 1, f(a)$$

If (m − 1) is not zero, then the second execution of the THEN component takes place. This removes the values from the top of the n-stack and places them (in reverse order) in the s-stack.

$$(s2, s1) = (f(a), m - 1)$$

(the s-stack is indexed from its base).

The iteration then causes computation of f(f(a)) and (m − 2) which appear on the top of the n-stack in order

$$(m - 2, f(f(a))$$

It is easily seen that the continuation of the procedure will result in the computation of

$$0, f^m(a)$$

for the top of the n-stack. At this point, the decision function, being n'', will be zero and the procedure will terminate, leaving only $f^m(a)$ at the top of the n-stack. The s-pointer will have been returned to the base at the end of the substitution operation.

For example, the iteration of multiplication is the power function.

↑ (n,) [DO, { THEN ⟨ −(x2, 1), *(a, x1) ⟩,
IF (x2) ; ; }] m, a

n-stack	s-stack
m, a	m, a
−(m, 1), a^2	−(m, 1), a^2

0, a^m

PRIMITIVE RECURSIVE FUNCTIONS. A primitive recursive function is one specified by a particular schema of computation. The schema is itself a function of functions. The function arguments of the primitive schema are the basis of the recursion, and the primitive function computed in this way is said to be "recursive in" these given functions.

The primitive schema is,

$$f(0, a) = b$$

$$f(n + 1, a) = g\,(n, a, f\,(n, a))$$

The variable n is the variable of recursion, and the function g is a given function. The function g itself may be a primitive function computed from still other given functions.

The realization of or the expression of this schema in SPEAK is now presented. Construct the function h

h: + (x1, 1)
x2
g (x1, x2, x3)

Then the substitution operation

{ h(x1, x2, x3) ; ; ; }

iterated with the stopping rule

N − x1

will result in the computation of the function f(N, a).

FIRST-ROOT OPERATORS. Given any function of two arguments,

f(n, m)

an operation may be constructed having for its value the first value of the argument n for which the condition

$$f(n, m) = 0$$

is satisfied.

This function, so defined for a fixed function f, is a number theoretic function of the argument m. Writing

$$\mu_i\,(f(i, m) = 0)$$

the function so defined will be called the first-root function. I is a functional since its value, if any, is dependent on the choice of the function f.

The variable i which appears in the formula above is bound. In SPEAK, bound variables appear only implicitly. This is simply expressible in SPEAK. In effect, the substitution operation is confined to the first variable argument,

{... f(x2, x1); ..;}

This has the effect of binding the variable x2 within the scope of the inner pair of brackets. The variable x1 is bound within the scope of the outer pair of brackets. There is no need for any further indication of which variable is bound and within what scope.

In general the first-root operation will have a primitive function as one of its arguments. The computation scheme already presented will be slightly modified by changing the stopping rule to

x3

This change will halt the iteration whenever the functional value, which is denoted by x3, becomes zero.

Of course, this may never occur. Whenever the function f(n, m) is such that one may assert that the following condition holds

i. E n (f(n, m) = 0)

then the first-root function represents a unique number. If, however, one may only assert that for some values of m, condition i. holds, then the function is known as a partial recursive function.

Every general recursive function may be expressed in the form

$$g(\mu_i(f(i, m) = 0))$$

in which the functions g and f are primitive recursive. Since SPEAK has the means for expressing the procedures for computation of a primitive function and a first-root operation, it has the means of expression to cover general recursive procedures.

EXAMPLES. In the following examples, it is to be understood that any and all of the computations expressed may be made in an interactive mode and on a term-by-term basis.

Example 1. The Desk-Calculator Mode

	n-stack
5 cr	5
+2 cr	7
−4 cr	3
*6 cr	18

This simple example indicates the method of calculating from the terminal keyboard. There is no effective limitation on the complexity of calculations that can be made in this way. In left-hand Polish the calculation just illustrated is

$$*(6, -(A, +(2, 5))) \text{ cr}$$

Note: cr is carriage return.

The next examples to be presented represent somewhat more complex calculations.

Example 2. The Fibonacci Sequence

This is a primitive function defined as follows:

$$FIB\ (0) = 0$$
$$FIB\ (n+1) = 1; \text{ if } n = 0$$
$$FIB\ (n+1) = FIB\ (n - 1) + FIB\ (n); \text{ for } n \neq 0$$

This function may be computed by iteration of the following substitution

$$\{+ (x1, x2), x2;;\}$$

with a stopping rule based on a count of the iterations. Tracing the computation, the successive versions of the s-stack and the n-stack are

n-stack	s-stack
1, 0	1, 0
1, 1	1, 1
2, 1	2, 1
3, 2	3, 2
. . .	. . .

Example 3. Greatest Common Divisor

As a final example of iterated substitutions, the routine for computing the gcd of two integers will be presented. The substitution operation to be iterated is

$$\{\text{MOD } (x2, x1), x2 ; ; \}$$

A trace of this operation, beginning with the arguments (12, 16) in the n-stack, is as follows:

n-stack	s-stack
12, 16	
4, 12	12, 16
0, 4	4, 12

The stopping rule is simply x2.

Note: The MOD function is the usual mod-function, the remainder after division of the first argument by the second.

Since the stopping rule contains the variable x2, the rule must be stated within the scope of the substitution operator

$$[\text{DO}, \{\text{THEN } (\text{MOD } (x2, x1), x2, \text{IF } (x2) ; ; \}]$$

This operation assumes that x1 is the smaller of (x1, x2). This condition may be assured by the following procedure.

Beginning with the pair 16, 12, at the top of the n-stack

$$\{\text{MIN } (x1, x2), \text{MAX } (x1, x2) ; ;\}$$

will place 12 at the top of the n-stack, with 16 just below.

The SPEAK Subsystem

The subsystem, in distinction to the language, consists of the program modules required to implement the language processor. The principal program modules and their relationships are indicated in figure II-3.

The monitor, which is central to the system, is intended to operate either as an independent free-standing monitor, or as a submonitor where the host system is a large time-sharing facility. Aside from the monitor itself, denoted by MONITO, there are four communications modules, two for communication with the disk, READ and WRITE, and two for communication with the terminal, TYPE and ACCEPT. The argument for a disk-read, or for a disk-write specifies a RAM file and a DISK file. The specification includes the number of storage units to be transferred. These four functions are machine dependent, and consequently are not specified in greater detail here.

INTERP. INTERP is the name of the interpreter. It is a table-driven automation. Table I is the classification table of the first 64 ASCII characters. Different classifications may be adopted. The classification may be extended to the full ASCII set, and the given classifications may be

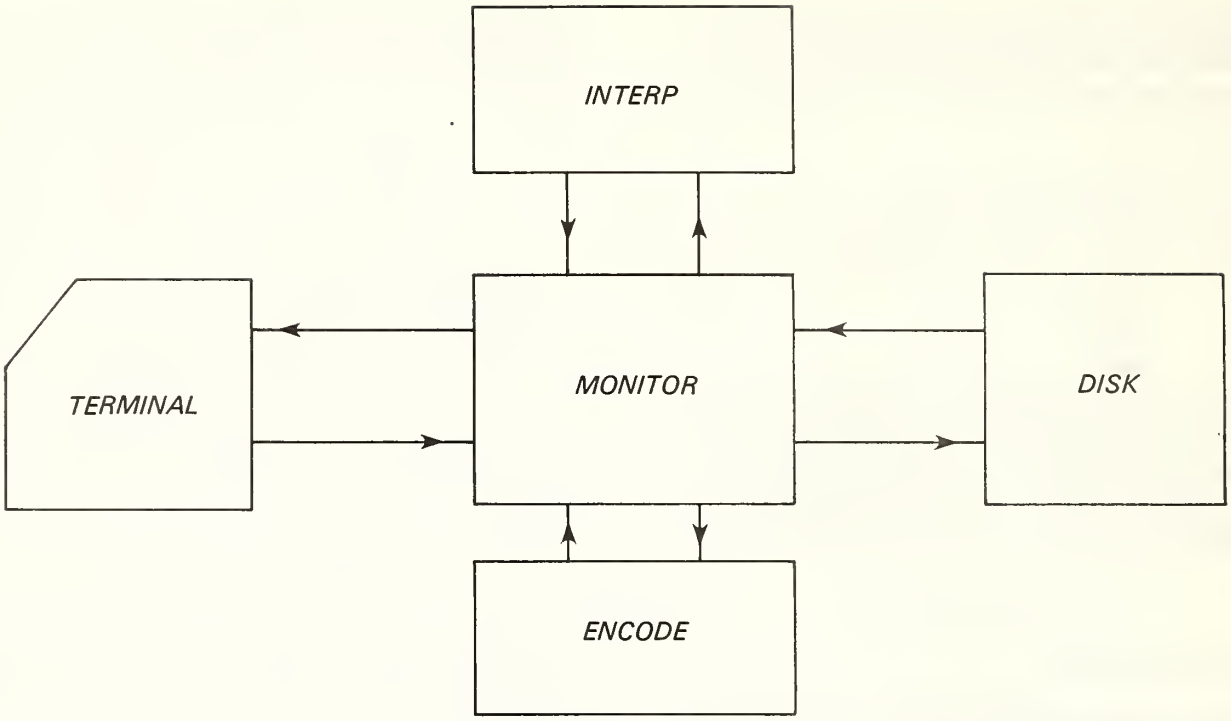


Figure II-3

subdivided. The following table is adequate for the purpose of demonstration.

Table I. Character Classification

0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
6	5	3	5	5	5	5	5	4	4	4	4	4	4	1	4
20	21	22	23	24	25	26	27	30	31	32	33	34	35	36	37
0	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4
40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57
4	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
50	51	52	53	54	55	56	57	60	61	62	63	64	65	66	67
2	2	2	2	2	2	2	2	2	2	2	4	4	4	4	4

In the interpreter, the input will be designated by the pointer, "ch" which will point to the octal representation of the input character, the upper integer of the pair, while the function CLASS (ch) refers to the lower integer of the pair in the table. The significance of the class marks is indicated in the following list; class 0, digits; class 1, decimal point; class 2, alphabetic character; class 4, arithmetic symbol; class 5, other characters; and class 6, the space.

The automaton has six states:

Table II. States

0—initial state
1—integer
2—fraction
3—alphanumeric
4—arithmetic symbol
5—literal

The state other than the initial state, is determined by the type of word being generated. Thus when an integer is being generated, the automation assumes the state-1. The automation always begins in the initial state and returns to this state at the completion of the formation of a word, or after the execution of an error interrupt.

Within the interpreter, states and classes will be referred to by code numbers, from 0 to 6 in the case of character classes, and from 0 to 5 in the case of states. Each input character will result in a recomputation of the class and state of the system. On this basis a new state, and some functional procedure will be selected. There are seven system functions that may be called from the main program module.

The symbol table contains the addresses of the interpreter functions associated with the following alphanumeric names. These functions are identified in the program by code numbers from 1 through 7.

Table III. Interpreter Subroutines

1: FRACTION
2: INTEGER
3: ENDNUM
4: ENDINT
5: STRING
6: INITIAL
7: ERROR

Integers are interpreted according to the system radix, a parameter. Powers of the radix are accumulated in the system address, "power."

Interpreter Subroutines

TITLE: FRACTION

ENTER:

power ← * (power, radix)
CALL (INTEGER)

RETURN:

TITLE: INTEGER

ENTER:

count ← + (count, 1)
IF (count GT 10) GO TO ERROR
int ← + (* (int, radix), −(ch, bias))

RETURN

Note: ch − the input character.
bias − the ASCII bias for a numerical digit (60 octal)
count − the character count of the current word
radix − the system radix (usually either octal or decimal)

TITLE: STRING

ENTER:

count ← + (count, 1)
IF (count GT 6) GO TO ERROR
DEPOSIT (ch)

RETURN

Note: DEPOSIT is an operation or a pseudo-operation for depositing characters in a string.

In addition to the functions FRACTION, INTEGER, and STRING, there are three formatting functions which take the outputs of the word-forming procedure and place them in SPEAK internal format.

These formatting functions are ENDFRACT, ENDINT, and ENDSTRING. ENDINT, and ENDFRACT, simply place the output word in a SPEAK word

(−1, integer)

TITLE: INTERPRET

ENTER:

ch ← + (ch, 1)
class ← CLASS (ch')
state ← STATE (state, class)
function ← FUNCTION (state, class)
JSR (function)

RETURN

Note: ch designates the character pointer.

ENDSTRING has two forms of output. If the word formed is a literal, then the word is placed unchanged in the output. However, if the alphanumeric word is not a literal, its definition is obtained from the symbol table.

Entries to the symbol table are set in by means of a DEFINE statement, which establishes the correspondance between the input word, its rank, and its numerical equivalent. Actual addresses are assigned by the B-tree handler which manages storage in general.

Table V is the driving table of the automation. This is a double entry table, entered by entries designating the state, and the class of the input character. The functions STATE (state, class) and CLASS (state, class) represent the entries in the driving table.

Table IV. Driving Table for Interpreter

CLASS	0	1	2	3	4	5	6
STATE							
0	2,1	,2	5,3	,5	5,4	,0	,0
1	2,1	,2	8,	8,	8,	8,	4,0
2	1,2	8,	8,	8,	8,	8,	3,0
3	5,3	8,	5,3	8,	8,	8,	6,0
4	8,	8,	8,	8,	5,4	8,	6,0
5	5,5	5,5	5,5	,0	5,5	5,5	5,5

Notes: Each table entry is a pair, TABLE (state, class) = (FUNCTION (state, class), STATE (state, class))

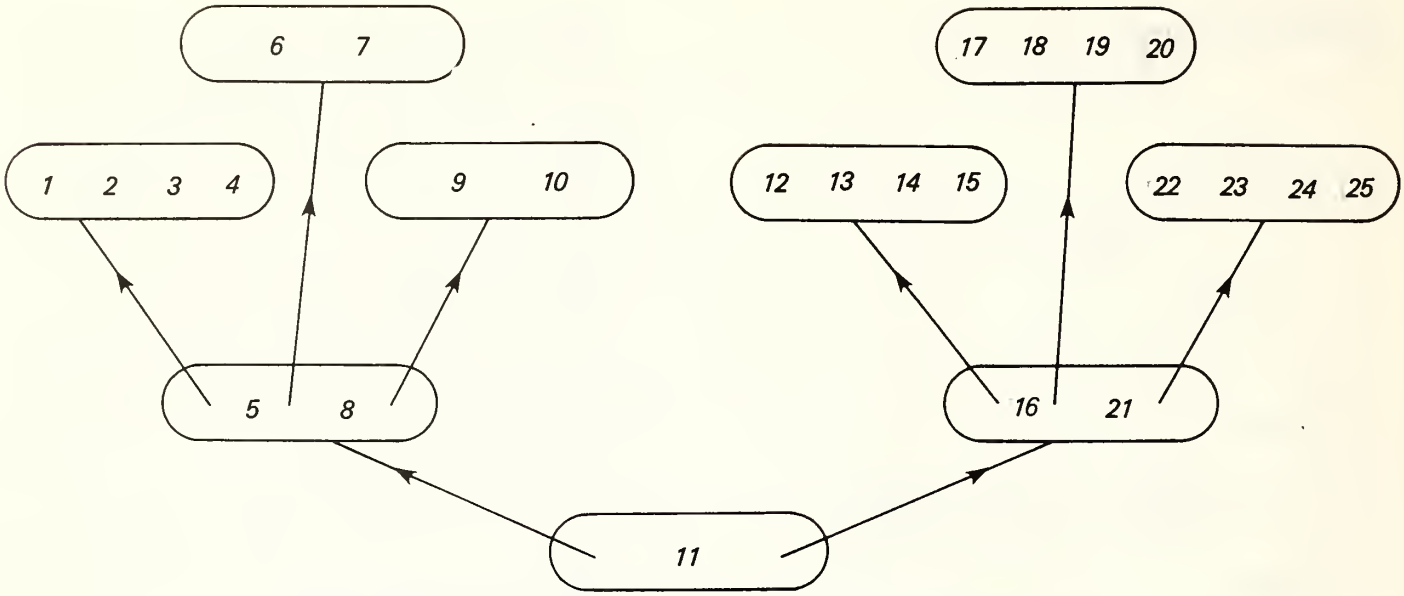
The Function ENCODE. The purpose of the ENCODE function is to interpret the internal words which are simply bit-strings, as strings of ASCII characters. The different interpretations allowed are ASCII, decimal, octal, and generally, any other radix whatever, such as binary, or hexadecimal.

Three things must be specified in the argument: a source file of data, a destination file, (often the terminal itself or some fixed buffer), and a format, which is a simple directive in the form of a list, from which the various interpretations are taken.

Thus,

(50, 5A, " " ,10D, "." 5D)

represents 50 records each consisting of five ASCII characters, followed by three spaces (specified by the literal designated by quotes), followed by an integer consisting of ten decimal digits, blank filled on the left, followed by a decimal point, followed by a decimal fraction, blank filled on the right.



Illustrative B-Tree — Root, two intermediate branches, and six leaves.

Figure II-4

The routines for performing these interpretations are:

ASCII
INTEGER
FRACTION

Where these procedure names are contextually distinguished from the identical names used in the program module, INTERP. A format is simply a list, stored under the name to be associated with the list. Thus, the function with its arguments is written as ENCODE (format, source, destination).

B-TREES. The primary method of storage organization of ARITHMICON is the B-tree. These trees are exactly those described by R. Bayer and E. McCreight[1] For reference, their definition is quoted here. The B-tree is a system of organization of the pages containing an ordered set of indexes. The tree is a collection of pages of index numbers and pointers for which each page is identified with a node of a rooted tree.

The individual record on each page is alternately pointers and data. In the present application, the data consists of two words, the index and a pointer to data associated with the index (this data being irrelevant for the purpose of the present discussion). On any page, the first and last elements are pointers to the ordered successors of the node (page).

The first such pointer points to the predecessor (father-node) of the page, and the remaining pointers point to successors (son-nodes) of the page.

In such a system, the pages are not ordinarily full. This allows for the insertion of data. When a page is full, the attempt to insert data in the page will require that the page be split. On the other hand, when two pages are half, less than half full, and are adjacent, sons, the pages may be consolidated into a single page.

Whenever the root page splits, the height of the tree is increased by one. When immediate successors of the root are concatenated, the height decreases by one.

Pages without successors are called leaves. Any path from the root page to a leaf has the same length measured in terms of intervening pages, as any other such path.

The formal constraints are:

1. Each path from the root to any leaf has the same length h, also called the height of the tree. The number of nodes from the root to a leaf, inclusive, is h.

2. Each node except the root and the leaf nodes has at least k+1 successors or sons. The root is either a leaf or has at least two sons.

3. Each node has at most (2 k + 1) sons.

Each page is provided with a header containing three words of information p_{-1}, j, n. The first word is a pointer to the father node. The second word denotes the order of precedence of the node among brothers. The third word denotes the number of keys in the page (n LE k).

$$p_0, y_1, p_1, \ldots, p_{n-1}, y_n, p_n$$

Each of the pointers, p_j, points to a son-node having the same data structure. If all the keys are blank, the node is a leaf.

[1] 12 Informatica, Vol. I.

Finding a Key in a B-tree. The procedure is expressed in SPEAK.

```
    { READ (x2, x1) ; }] root; buffer }
    (μi (y (i) GE key)
    IF (y (x3) ≠ key) ; }
  { THEN < p (x3 – 1) >
{ [ DO
```

The steps of this operation will be traced. The successive lines are read from right to left, since the notation is left-hand Polish. The opening brace indicates the start of a substitution. The word, buffer, is then stored by the semicolon. The word root is then placed at the top of the n-stack. Note, at this time, the word, buffer, is in the top of the s-stack.

The instruction READ (x2, x1) for these values of the variables, namely root, and buffer, reads the root page into the buffer designated.

The next operation is a first root operator which finds the first index for which the stated condition is satisfied; this index is placed in the top of the n-stack.

The IF operation contains the variable, x3, which is bound by the substitution appearing at the end of the line. The effect of this substitution is to bind x3 to the value of the first root just computed. This substitution operation removes the first root from the n-stack and places it in the substitute stack. The IF operation places either a 1 or a zero in the f-stack. The THEN operation will be executed if the condition tested above is satisfied. This operation puts the pointer indicated in the top of the n-stack. The DO operation causes iteration of the procedure. Note that the left braces as encountered, clear the s-stack of the substitutions, unbinding the variables. For example, the final left brace clears the s-stack of the word, buffer.

In defining first-root operations, such as the one used here, it is customary to define the value of the root for those cases in which no index can be found satisfying the condition. In this case, the required value is simply the fictitious index, n + 1. In this case the pointer value computed will be p_n, the final pointer of the page.

Reading Keys in Sequence.

The successor relation. The relation between a key and its successor in a B-tree may be expressed in SPEAK. The expressions describe the sequential construction of chains of pointers.

The upward chain. To find the successor of a key in a given node, not a leaf node, the following chain is constructed,

```
    { READ (x1, x2) ; }p(k) ; buffer }
    IF (p'0 ≠ 0),
    THEN (p(0))
{ [ DO
```

In this sequence, the pointer p_k is read in the original buffer contents, placed in the n-stack, and the iterated substitution performed until a value of $p'_0 = 0$ is attained. At this point, the value of y_1 is the successor.

The downward chain. To find the successor of a key in a leaf-node one merely takes the next key in sequence within the leaf if such exists. If the key is the last in the leaf, its successor is found by constructing the downward chain, using the values of p_{-1}, r, and n, from the header information.

```
    READ (x2, x1) ; r ; p-1 }] ; buffer }
    { IF (x3 = n)
{ [ DO
```

Tracing the operation; buffer is first set as the argument, x1, p_{-1}, as that of x2 and r, that of x3. The READ instruction brings the new page into buffer. The IF instruction compares x3 with n (from the header on the new page). If the operation is iterated, the new p_{-1} and r are copies from page into the n-stack.

Actual insertion, deletion, and updating of header data is carried out in a text editor. The text editor also performs the page or node-splitting operation, and it also consolidates adjacent brothers when this is possible.

The Subsystem TOPO

A geographic base file (GBF) is a collection of subfiles containing descriptive properties of the discrete geometric objects known as cells. There are therefore three types from the point of view of dimension, 0-, 1-, and 2-cell descriptive files.

Thus, given a cell identifier, one may enter the descriptive file with the cell identifier, and obtain the description of the cell. Conversely, one may construct index files, which are representations of the lists of cells having a given descriptive property. One may therefore obtain from the file system a list of cells having some stated property or combination of properties represented by the basic data.

This part of the system is quite customary. However, the fact that among the descriptive properties, the property of incidence is singled out means that one can obtain not only those cells having a given property, but in addition, one may retrieve from the file the complete geometric description of the cellular object by the chosen description.

Some examples are: the collection of 2-cells making up an administrative district, the collection of 1-cells bearing a given descriptive street name, the collection of 0-cells for which a specified event has been recorded.

In part I, the basic data retrieval functions have been defined. Central to this system is the file called inlist, the list of segments present in the immediate memory. From this list, one obtains the internal list, toplist, and any further descriptive segment data required for a particular application. By using the HOMOL function, one obtains lists of incident cells, either 0- or 2-dimensional, and from these lists one may obtain access to the related descriptive property files.

The descriptor files are identified by dimension of the geometric object described, and the particular descriptive property. Index files based on these descriptions are essentially lists of all cells having a property identified by the descriptor.

In figure II-5, the method of retrieving a complete description of a specified geometric object is illustrated. In the first example, it will be assumed that the geometric structure is defined by a list of segments (1-cells) contained in inlist. The function, DIME, retrieves the topological data and places it in toplist. The one-dimensional descriptors may then be consulted to obtain any required descriptive data relating to the segments in inlist. Next, by using the HOMOL function one obtains the list of incident cells, 0-cells and 2-cells, and from these lists one obtains the descriptors of these cells. As shown in the illustration, two applications of the HOMOL function are required for this purpose.

By methods presented in Part I, it is possible to expand the neighborhood of cells for which the description is to be retrieved. One of the important advantages of ARITHMICON is that this expansion of the neighborhood may be accomplished by interactive direction.

The second situation to be considered is that in which one begins with some stated property. Entering an index with the specified property will fill the list, cellist, with the cell identifiers corresponding to the property. By use of the E1 operation, one obtains the list of incident 1-cells. These appear in seglist. The function, SELECT, transfers designated cells to inlist, and the description of the defined object may be obtained by the procedure outlined in the preceding paragraphs.

For many purposes, such as the administrative use of a geographic file, the data retrieval system just described is adequate. However the construction of a large geographic file by any of the techniques currently available is a difficult undertaking, and it is unlikely that one can obtain by these means an unedited file of adequate quality. One should be prepared to edit any newly prepared geographic file. The procedures for conducting such an edit are those described in part I.

Editing Procedures

Two kinds of editing procedures are used to establish the topological and metrical consistency of the map model. These procedures are designed to follow exactly the requirements of the map editing procedures of part I.

Every 2-cell of the model is assumed to be interpretable as a 2-cell in some geometric object. The generalization of 2-cell to include the case of disks from which interior 2-cells have been removed is essentially a trivial one, introduced only for convenience. The 2-cell edit is simply a test that the boundary of the cell is a simple 1-circuit, or a set of components consisting of sets of 1-circuits that can be interpreted as boundaries of holes in a disk.

The second set of procedures examines the fundamental neighborhood of each 0-cell interior to, or on the boundary of, a closed fundamental neighborhood of a given 0-cell. It is required that each of these neighborhoods satisfies the "cocycle" condition for interior points, and for points on the boundary, the incident 1-cells of a coboundary must form a cochain beginning and ending with a boundary 1-cell.

These edits are carried out by means of procedures detailed in the sequel.

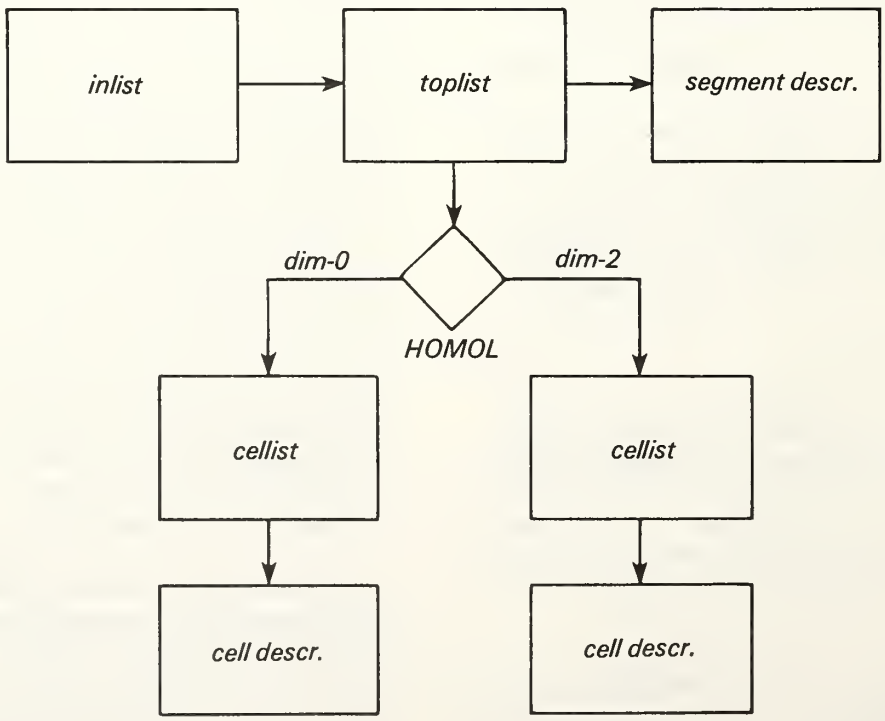


Figure II-5

EDITING A 2-CELL. The sequence in SPEAK,

SET (cellist, cell), 2
E1 (seglist,
SELECT (coeff ≠ 0, inlist,
DIME (toplist,
CHAINS (chainlist,
KIRCHOFF (codelist,

will cause the retrieval and analysis of the boundary of a 2-cell. For an absolutely regular 2-cell, that is, a disk without holes, this boundary code should be

1; 1, 1, 1, 0

a single component consisting of a loop.

If any different code is obtained, an extended analysis must be made. However, in the case the correct code is returned, the procedure

CLEAR n

will clear the n-stack for a new analysis.

The procedure

INTERSECT (chain)

will be executed. This will return a decision outcome (0 or 1) to the n-stack, according as there are no intersections of the chain with itself, or as there is at least one such intersection. In the latter case, the interactive editing program, DISPLAY will be initiated.

Any topological defect of the 2-cell boundary, such as the presence of acyclic elements, will also be referred to the program, DISPLAY.

In case there are multiple components of the boundary, and the components are all cyclic, the external boundary must be identified. The procedure,

EXTERNAL (chainlist)

will return the identifier of the external boundary chain to the top of the n-stack. If no chain is external an error return is made to he interactive program, DISPLAY.

Having identified the external boundary, the iteration of the procedure of obtaining the homologous internal boundary will ultimately reach a state at which the boundary is homologous to zero. See appendix A, part I.

At each stage of this procedure of boundary analysis, the circuits are tested against each other and against themselves for self intersections,

INTERSECT (circuit-i, circuit-j)

All error returns will be made to the DISPLAY procedure.

The end result of this editing procedure is to verify that the boundary of the 2-cell is a simple loop, after filling all of the holes in the disk. Moreover, all of the hole-boundaries and the disk boundary itself are topologically and metrically consistent.

EDITING A 0-CELL

The following SPEAK sequence will retrieve the coboundary of the 0-cell.

SET (cellist, cell, 2)
E1 (seglist,
SELECT (ALL, inlist,
DIME (toplist,
CHAINS (chainlist,
KIRCHOFF (codelist

At this point any code different from (1, 1, 1, 0) will cause an error return to DISPLAY.

After any necessary corrections, the fundamental neighborhood must be formed. This is done by first taking the boundary of this neighborhood.

CLEAR n
HOMOL (cellist, 2, left, toplist)
E1 (seglist, 2,
SELECT (coeff ≠ 0, inlist
DIME (toplist,
CHAINS (chainlist,
KIRCHOFF (codelist,

From this point on, the same procedures used in analyzing the 2-cell will be followed. The holes, if any, in the disk forming the fundamental neighborhood will be filled, and any repeated segments in the disk boundary will cause an error return to DISPLAY.

The principles of this edit may be summarized quite simply.

The coboundary of the vertex is retrieved, together with the two-dimensional coboundary of this coboundary. The usual cocycle condition is verified.

Next, if the boundary of this coboundary is either in components, or has multiple loops, the external boundary of the disk is identified, and the remaining boundary circuits are homologous to the external boundary. By taking the negatives of these circuits, and iterating the procedure of taking the internal coboundary cells, a point will be reached at which the boundary is null, that is to say, all holes have been filled and the internal boundary is null, or the external boundary is homologous to zero.

The remaining tests are metric. The vertex must be positioned internal to its fundamental neighborhood. A PIP procedure for the point relative to the external boundary will verify this condition.

In determining the external boundary, two tests are made. First, the procedure, INDEX, is used to determine the angular rotation of the successive edges of the boundary polygon during a complete traverse. The index must be either +1, or −1. The positive sense is taken to be anticlockwise rotation.

The external boundary should have index, +1, and the internal boundaries should be of index, −1.

Any error detected at any stage causes an interrupt to the DISPLAY program.

The Subsystem GRAPH (Components and Functions)

The subsystem, GRAPH, is designed to control the interactive graphics terminal, and to provide all necessary geometric support for the display and manipulation of graphic images required for the editing of maps. The commands of GRAPH will be expressed in the SPEAK language.

The graphic terminal operates in two modes. In the first, iconographic, numerical models are converted to images on the face of a cathode ray tube. In the second mode, iconoscopic, or image viewing mode, the present state of the art does not support a true image-viewing mode. This mode is usually simulated by such analog devices as light-pens, mechanically conditioned cursors, and stylus-tablet devices. The elementary iconoscopic element "viewed" by these means is the point. All references to an image are transmitted by the transmission of textual data, and the procedure of point-identification, pointing.

When an observer points to provide positional signals, the act of pointing is denoted by a text symbol, #. When the language processor encounters this symbol, an interrupt is generated. This interrupt is processed by displaying of the cursor, and a wait state which requires a pointer positioning, and return indication from the observer. Thus any expression such as,

DIST (#, #)

represents a function of arguments to be furnished by the observer by pointing to positions in the image space. In the particular example cited, the function is the distance function, and the result of executing this command would be to place the distance between the designated points on the top of the n-stack.

Since the actual procedures required for the module, GRAPH, require the implementation of a number of lower-level geometric procedures, some simple examples will be given. These examples, at the same time, will provide some familiarity with the mode of expression in SPEAK.

GEOMETRIC ROUTINES. In geometric routines, the storage units will be the point and the segment. A point will be designated by a pointer to the first of two coordinate values. A segment will be designated by a pointer to the first of four coordinate values. The segment will be considered to be directed by the order of the coordinates as stored. The direction will be assumed, from the point (u1, y1) to the point, (u2, v2).

The distance function, DIST (p1, p2) will be computed from the procedure,

```
* ( – (u2, u1), –(u2, u1))))
* ( – (v2, v1), –(v2, v1))
SQRT ( + (
```

Without going into detail unnecessarily, it will be assumed that the usual procedures for computing inner products and vector products are available as executable subroutines in NATIVE. The routines will be described as they are introduced.

INTERACTIVE RULING. The cursor may be used to rule a rectangle in the image space. Setting the horizontal cursor at the base of the rectangle, the horizontal limits may be indicated by the vertical cursor, with two data transmissions. The horizontal cursor is then reset to the top of the rectangle, and the horizontal limits set equal to those previously established. The cursors function in this way as T-squares.

PLOTTING. Since plotting commands differ from one plotter to the next, the plotting modes described here are those of the FR-80.

There are seven plot commands, (inclusive of the null command). These are, M, D, DM, MR, DR, DMR.

These commands are interpreted in the following way. The last position of the beam (or the current position) is designated by cp. Execution of an M command always moves the position of the cp to an updated position designated by the command argument. Thus

M (#)

moves the plotter beam to the position indicated by the pointer. On the other hand the command,

DM (#)

will draw a vector from the position, cp, to the position indicated by the pointer, #, and will update the cp pointer to the position, #. The command D will simply draw a vector from the cp to the point indicated by the argument.

The sign, R, denotes relative computation of the position. In other words, the position indicated by the argument is computed relative to the cp, as the vector sum of the cp coordinates and the coordinates of the argument. The addition of the R as a suffix increases the number of non-null commands to six.

The particular plotter referred to, the FR-80, also provides for the use of a repeat command. This will be ignored in this presentation. Repetitive sequences of plot commands may be expressed in SPEAK.

The two modes of plotting designated by D and DM correspond to the topological notions of chain and cochain.

For example, a sequence of D commands,

D p_i

where the p_i are points of a list, will result in a plot of a star of rays emenating from the position marking the current cp.

On the other hand a command, M (p_1) followed by a sequence

DM (p_i) i = 2, . . . ,

will result in the plotting of a connected chain, a polygonal arc, beginning with the first point of the list, and ending with the last.

ICONOSCOPIC FUNCTIONS. These functions, by means of which the terminal "views" the image, are fundamental to the system of interactive graphics. They are designated as NRPOINT, NRSEG, and NRBLOCK.

As suggested by the mnemonics, NRPOINT finds the point of the image nearest to the designated point, NRSEG finds the segment nearest the indicated point, and NRBLOCK finds the identifier of the 2-cell in which the point is located. This last function is implemented by determining the half-plane in which the designated point lies relative to the nearest segment.

An analogous function NEARCH, will identify the nearest character to the position pointed to.

Coordinate Transformations. The internal map model is furnished with a coordinate system known as the model coordinates, or the model frame of reference. The image space on the face of the cathode ray tube (CRT) is referenced by a coordinate system known as the raster coordinates. The schematic diagram in figure II-6 illustrates the relations among these coordinate systems and the actual image. There are three distinct coordinate systems to be considered: those of the model, the raster coordinates, and the actual geometric coordinates of the tube face. For all practical purposes, the distinction between the coordinates of the tube face and the raster coordinates may be ignored, and these systems will be referred to only as the raster system.

The procedure scale operates on a segment list. With a given image space designated by u_{max}, u_{min}, v_{max}, v_{min}, and a given model image with range, x_{max}, x_{min}, y_{max}, y_{min}, the SCALE procedure determines the parameters of a similarity transformation which results in the largest image of the model that will fit in the image space. These parameters are a translation of the center of the model image, relative to the center of the image space, and a scale factor. The inverse of this transformation is computed simply.

The procedure, SCALE, is applied to the segment list for which the graph is required. The actual raster coordinates are held in a list corresponding to the segment list (inlist). This list is known as "plotlist."

A second scaling procedure is used to obtain an enlarged image of a specified neighborhood of the map. The procedure is called ZOOM, because of the analyogous photographic procedure. ZOOM parameters are determined with reference to a given image. The cursors are used to define the area to be displayed. This requires four data transmissions from the cursor. Two transmissions define the horizontal limits, and two define the vertical limits of the image.

A procedure, CLIP, is then applied using the new image parameters. CLIP computes the intercepts of segments belonging to the rescaled raster image, and eliminates those parts of segments outside the image space.

The relationships among the several image models are indicated in figure II-7.

THE INTERACTIVE EDIT DISPLA. The program DISPLA is the main control for the analysis, correction, and updating of the map model.

The illustration shows the image space as seen from the crt. DISPLA is supported by a number of lower level modules: NEARPOINT, NEARSEG, NEARBLOCK, NEARCH.

A number of single characters have a contextually defined meaning. The first group of such characters is: M-map, N-name, V-node, B-block, T-tract, H-house. These characters are interpreted within the module DISPLA. Their meanings are defined only within this context. Thus,

M (#)

will result in a typeout at the indicated position of the map-sheet number of the nearest vertex. Any of the characters listed above followed by the sign, #, will result in an annotation of a reference graph.

For example, N (#) would annotate a segment according to a fixed format with the associated street name.

If a command such as V (#) is executed with the pointer outside the image space, all elements of type V (in this case nodes) will be annotated.

The second group of characters is interpreted within DISPLA as calls to corresponding subroutines. The characters of this group are, U, C, D, M, Z, MV, W, R, EX, HOM, R.

These characters or character groups are interpreted as subroutines.

The character E is a qualifier for the graph annotation procedures, signifying that only essential vertices are to be annotated.

The functions designated are as follows:

U is the update procedure. This procedure has been described within the section dealing with the subsystem, TOPO.

Correction data may be supplied by using the DIME format as the format of the correction data.

Thus

segment, node, node, block, block

will be interpreted as the new segment to be inserted. The old segment will in effect be deleted, and the new one inserted.

For each nonblank field, the update procedure is performed, following the insertion of this data and the character, U (for update).

In the mode, change, designated by the character C, one begins by designating a segment with the cursor. The cursor disappears and reappears at the from-node of the segment. A typein (including a blank) is accepted and the cursor disappears and reappears at the to-node. The cursor moves through the DIME sequence, covering all four segment fields.

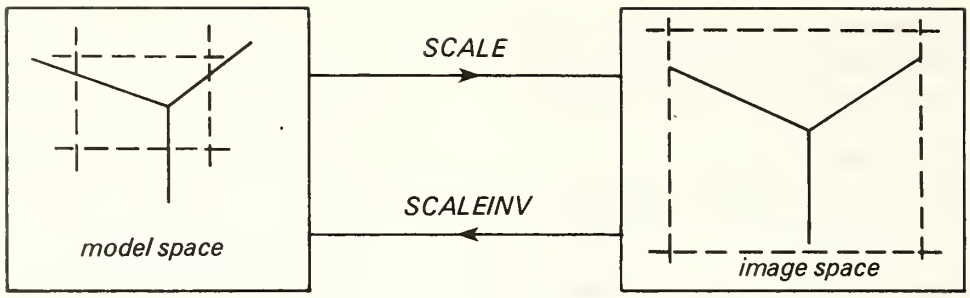


Figure II-6

CELL 10 101
DIMENSION 2
COMP. n1 n0 μ α
1 1 2 0 1

SET CROSS HAIR, ENTER CHAR:

. . . . . .

N V B E S C A Z D N R W #

SET CURSOR, THEN ENTER

M – MAP
N – NAME
V – NODE
B – BLOCK
T – TRACT
H – HOUSE
U – PERFORM UPDATE
E – ESSENTIAL
C – CHANGE
Z – ZOOM
D – DELETE
MV – MOVE
W – WALK
R – RETURN

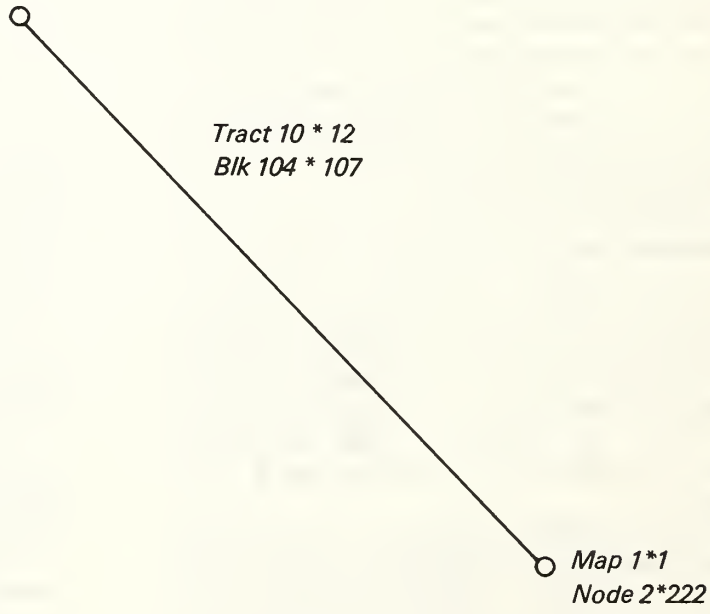


Tableau of CRT screen, DISPLAY

Example illustrates changes in Tract, Block, Map, and Node numbers.
Graph consists of the single segment as shown

Figure II-7

With the data being put in this way, the symbol U will cause execution of the update program. The reason for moving the cursor is to cue the observer as to the proper orientation of the segment. The data is typed in a single line, as shown in the standard format above. There is, therefore, no overwriting of the annotation in the image space.

The DISPLA program also functions in the analytic sequences. For example, if a fundamental neighborhood of a vertex has a boundary not homologous to zero, then the procedure, EX, will find the external boundary, identifying the chain. HOM is then used to obtain the homologous internal boundaries. These boundaries are replaced by other homologous boundaries forming a chain of homologous boundaries, finally terminating with the original external boundary homologous to zero. This chain of homologies may be followed graphically through the display program.

A segment may be deleted entirely by the command,

D (#)

in which the sign, #, designates the segment to be deleted.

A new segment may be created by the command,

CS

The DISPLA program will respond with an assignment of the segment identifier,

CS segid

The observer will then fill the remaining fields in order, and then an update command will be issued to insert the new segment in the file.

Coordinate Corrections. The procedure

M (#, #)

will cause the movement of the point from the first to the second indicated position, counting from the right. That is, one first identifies the point, then indicates the position to which it is to be moved.

The movement of a point is accompanied by a change in the file coordinates associated with the point. The procedure is facilitated by use of the ZOOM procedure, which will provide an enlargement of the image in the neighborhood of the point to which the specified point is to be moved.

Summary of ARITHMICON Procedures

SPEAK

PARSE	; the pushdown parser, from p-stack to n-stack
IF	; transfers the decision from the n-stack to the f-stack.
THEN	; conditional parse, if stopping rule on f-stack.
ELSE	; identical with above for complementary stopping rule
ENDIF	; removes governing decision from f-stack
DO	; conditional parse beginning at reflexive reference.
[ ]	; scope of a reflexive reference, beginning and end
{ }	; scope of a substitution
⟨ ⟩	; conditional string
↑	; pop stack
↓	; push to stack
←	; transfer data to address
;	; transfer substitute values to substitution stack
+, –, *, **, /	; usual arithmetic operators.

Note: SPEAK procedures written in left-hand Polish; parentheses and commas are ignored by the processor, and may be used as desired.

AUXILIARY FUNCTIONS

READ	; transfers a page from a disk file to buffer
WRITE	; transfers a page from core to a disk-file
TYPE	; types an ASCII character string to terminal
ACCEPT	; accepts an ASCII string from terminal
INTERPRET	; interprets input character strings
ENCODE	; encodes internal formats to ASCII for output
BFIND	; finds a key in a B-tree
BINSERT	; inserts a key in a B-tree
BDELETE	; deletes a key from a B-tree
BSEQ	; obtains a sequence of keys from a B-tree
UPDATE	; replaces a DIME segment and associated index references by a new segment with new index references.

TOPO

CELL	; lists input in cellist
E1	; retrieves segments incident to cells in cellist
SELECT	; selects segments to be processed
DIME	; retrieves topo data for selected segments
HOMOL	; transfers selected cells to cellist
CHAINS	; forms essential chains from a segment list
KIRCHOFF	; connectivity analysis of a graph
INTERSECT	; tests intersection of two chains
PIP	; point in polygon routine
INDEX	; positive or negative sense of traverse of a boundary
EXTERNAL	; identifies an external boundary
FOLDS	; detects the presence of folds in a neighborhood

GRAPH

NEARPOINT	; gets nearest point to cross hairs
NEARSEG	; gets nearest segment
NEARBLOCK	; gets nearest block
NEARCHAR	; gets nearest character
PLOT	; plot graph from plotlist
SCALE	; scales a model image to crt image space
SCALEINV	; inverse of scale
CLIP	; intercepts of segments with borders of image space
ZOOM	; magnifies local neighborhood
WALK	; a version of the psuedo dual is plotted
DISPLAY	; display codes listed below
N	; name of street
H	; house number
V	; node number
M	; map sheet number
T	; tract number
B	; block number
E	; essential
C	; change
D	; delete
MV	; move
W	; a call to WALK
Z	; a call to ZOOM
R	; RETURN
U	; call UPDATE

Appendix A. Homology

The notion of a homology can be formulated in terms of cells, chains, and the boundary and coboundary operations. A 1-chain is said to be homologous to zero,

$$C^1 \sim 0$$

if it is the boundary of some 2-chain. In this case, the chain must be some collection of 1-circuits. The term "cohomology" may be defined by replacing the elements of the above definition by their duals.

Two 1-chains are said to be homologous to each other

$$C_1^1 \sim C_2^1$$

if

$$C_1^1 + C_2^1 \sim 0$$

Since the bounding 1-circuits are all homologous to zero, every 1-circuit satisfies some homology

$$C^1 \sim \Sigma\, x_i N_i^1$$

in which the terms, N_i^1, form a complete set of nonbounding 1-circuits.

A few examples will help to clarify these notions.

Figure A-1 represents the map of a torus on a rectangle. The two independent nonbounding 1-circuits are represented by a pair of adjacent edges, a and b, as shown. Every 1-circuit on the torus satisfies a homology of the form

$$C^1 \sim x_1\, a + x_2\, b$$

For example, the 1-circuit C^1 indicated on the diagram, together with the nonbounding circuits b + 2 a form the boundaries of two 2-cells, and hence this combination is homologous to zero. It follows that the particular C^1 is homologous (mod 2) to the 1-circuit b.

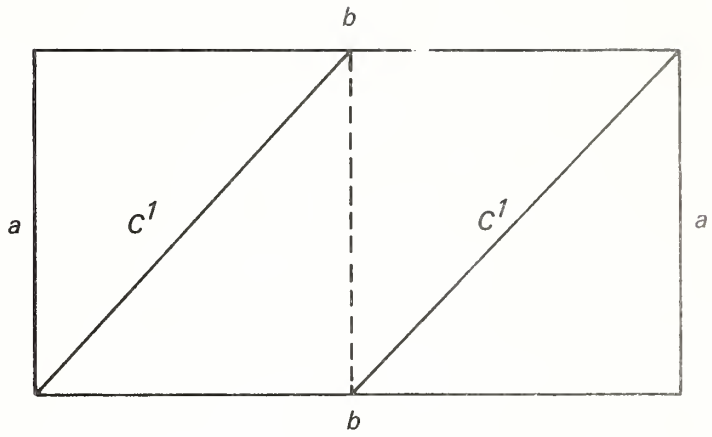


Figure A-1

The second example is the annular ring of figure A-2. Each of the pair of 1-circuits a and b, which together bound the annulus, are homologous to one another, but neither is homologous to zero.

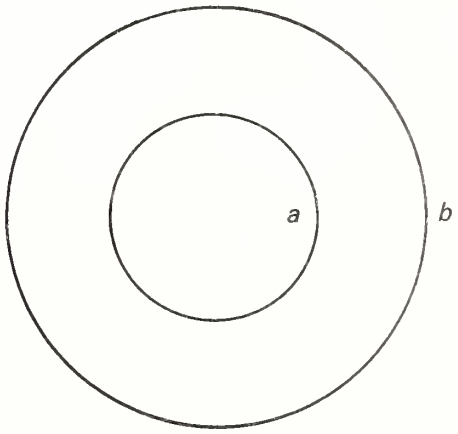


Figure A-2

Appendix B. Syntax of SPEAK

In the presentation of substitution already given, the bound variables were named sequentially, as x1, . . ,xn. This was done in the interest of simplicity in explaining the operation of the substitution stack. A freer use of variable names may be achieved by a simple modification of this operation, which in effect subscripts the substitution sign, ";", thus,

; variable-name, procedure

In the following discussion, the formal syntax of SPEAK will be presented as a sequence of productions. The convention for definition of a syntactic construct is,

construct being defined; definition

Literals are enclosed in quotes, and "term" is the construct already defined.

The highest level construct defined is the "procedure."

procedure: a sequence of terms, or
a substitution-procedure
an iteration, or
a conditional procedure, or
null

substitution procedure:

" { " procedure, substitution-sequence, " }"

iteration:

"[DO THEN", procedure, "IF", procedure, "]"

conditional procedure:

"ENDIF ELSE", procedure, "THEN", procedure, "IF", procedure

An eligible variable-name is an alphanumeric word not in the symbol table, and not bound in the substitution stack. A variable not bound is called free.

Syntax Errors

The principal errors detected by the processor are the following:

- Attempt to bind a bound variable
- Reference to a free variable within a procedure
- Ineligible variable name
- Unmated pairs of square brackets, braces, or "ENDIF-IF" pairs.
- Insufficient number of arguments for an executable procedure on the output stack (n-stack).
- Attempt to pop an empty system stack

☆U.S. GOVERNMENT PRINTING OFFICE: 1980-311-046/1142

working papers

The titles listed below are available in printed copies or on microfiche. The date in parentheses following the title indicates year of publication. Copies of recent papers may be reviewed and/or ordered at district offices of the Department of Commerce located in principal cities throughout the United States. Additional information, including prices, may be obtained by writing to Subscriber Services (ASD), Bureau of the Census, Washington, D.C. 20233.

No. 1. Raw Materials in the United States Economy: 1900-52 (Preliminary) (1954)

No. 2. The Role of the 1954 Census of Manufactures in Overcoming Problems of Industry Data (1956)

No. 3. Tract Data Compared for a 25-Percent Sample and a Complete Census (1956)

No. 4. Sampling in the 1950 Census of Population and Housing (1956)

No. 5. Occupational Trends in the United States: 1900 to 1950 (1958)

No. 6. Raw Materials in the United States Economy (1964)

No. 7. Papers Presented at the Census Tract Conference, December 29, 1958 (1959)

No. 8. Materials on the Preparation and Conduct of the U.S.S.R. All-Union Population Census of 1959 (1959)

No. 9. Historical Comparability of Census Manufactures Industries: 1929-1958 (1959)

No. 10. Papers Presented at the Census Tract Conference, December 29, 1959 (1960)

No. 11. Papers Presented at the Census Tract Conference, August 25, 1960 (1960)

No. 12. Papers Presented at the Census Tract Conference, August 28, 1961 (1962)

No. 13. Papers Presented at the Census Tract Conference, September 8, 1962 (1962)

No. 14. The Spectral Analysis of Economic Time Series (1963)

No. 15. Methodology and Scores of Socioeconomic Status (1963)

No. 16. Procedural Report on the 1960 Censuses of Population and Housing (1963)

No. 17. Papers Presented at the Census Tract Conference, September 5, 1963 (1964)

No. 18. The Measurement of Performance Potential in Manufacturing Establishments (1965)

No. 19. Tests of Use of Post Office Resources to Improve Coverage of Censuses (1965)

No. 20. Industry Classification and Sector Measures of Industrial Production (1965)

No. 21. A Spectral Study of "Overadjustment" for Seasonality (1965)

No. 22. Papers Presented at the Census Tract Conference, December 29, 1964 (1965)

No. 23. Spectral Analysis and Parametric Methods for Seasonal Adjustment of Economic Time Series (1965)

No. 24. Self-Enumeration as a Method for the 1970 Census of Housing (1966)

No. 25. Measuring the Quality of Housing (1967)

No. 26. Changes in the Structure of Manufacturing Employment (1968)

No. 27. Methodology of Consumer Expenditures Survey (1968)

No. 28. Metropolitan Area Definition (1968)

No. 29. Survey Applications of Social Psychological Questions (1969)

No. 30. Raw Materials in the United States Economy: 1900-1966 (1970)

No. 31. Price Variation in New FHA Houses: 1959-1961 (1971)

No. 32. Pretests and Dress Rehearsals of the 1970 Census of Population and Housing: A Procedural History (1972)

No. 33. An Estimate of A Quasi-Stable Age-Sex Distribution for Ghana in 1960 (1972)

No. 34. Investigation of Census Bureau Interviewer Characteristics, Performance, and Attitudes: A Summary (1972)

No. 35. Raw Materials in the United States Economy: 1900-1960 (1972)

No. 36. Response Variance in the Current Population Survey (1972)

No. 37. Who's Home When (1973)

No. 38. Economic Censuses of the United States: Historical Development (1973)

No. 39. Population and Housing Inquiries in U.S. Decennial Censuses, 1790-1970 (1973)

No. 40. Studies in Occupational and Industrial Classification (1974)

No. 41. Information Privacy and Statistics: A Topical Bibliography (1978)

Superintendent of Documents
U.S. Government Printing Office
Washington, D.C. 20402

Official Business

Postage and F
PENN STATE UNIVERSITY LIBRARIES
A00007096230?
COM-202
U.S.MAIL

technical papers

The titles listed below are available in printed copies or on microfiche. The date in parentheses following the title indicates year of publication. Copies of recent papers may be reviewed and/or ordered at district offices of the Department of Commerce located in principal cities throughout the United States. Additional information, including prices, may be obtained by writing to Subscriber Services (ASD), Bureau of the Census, Washington, D.C. 20233.

No. 1. The Sample Survey of Retail Stores: A Report on Methodology (1953)

... Description of the Sample for the Monthly Retail Trade Report (unnumbered revision) (1955)

No. 2. Accuracy of Census Statistics With and Without Sampling (1960)

No. 3. Farm Population, 1880-1950 (1960)

No. 4. The Post Enumeration Survey: 1950 (1960)

No. 5. Tests and Revisions of Bureau of the Census Methods of Seasonal Adjustments (1961)

No. 6. The Current Population Survey Reinterview Program, Some Notes and Discussion (1963)

No. 7. The Current Population Survey, A Report on Methodology (1963)

No. 8. Trends in the Income of Families and Persons in the United States: 1947-1960 (1963)

No. 9. Reconciliation of the 1958 Census of Retail Trade with the Monthly Retail Trade Report (1963)

No. 10. Population Trends in the United States: 1900 to 1960 (1964)

No. 11. Response Errors in Collection of Expenditures Data by Household Interviews: An Experimental Study (1965)

No. 12. Estimating Trading-Day Variation in Monthly Economic Time Series (1965)

No. 13. Sampling Application in Censuses of Population and Housing (1965)

No. 14. The International Standard Industrial Classification and the U.S. Standard Industrial Classification (1965)

No. 15. The X-11 Variant of the Census Method II Seasonal Adjustment Program (1967)

No. 16. Present Value of Estimated Lifetime Earnings (1967)

No. 17. Trends in the Income of Families and Persons in the United States: 1947 to 1964 (1967)

No. 18. Changes Between the 1950 and 1960 Occupation and Industry Classification (1968)

No. 19. The Current Population Survey Reinterview Program, January 1961 through December 1966 (1968)

No. 20. Correlation Between United States and International Standard Industrial Classifications (1969)

No. 21. Characteristics of America's Engineers and Scientists: 1960 and 1962 (1969)

No. 22. Measures of Overlap of Income Distributions of White and Negro Families in the United States (1970)

No. 23. The Position of United States in World Commodity Exports in 1968 (1970)

No. 24. The Annual Survey of Manufactures: A Report on Methodology (1971)

No. 25. Demographic Computer Library (1971)

No. 26. 1970 Occupation and Industry Classification System Showing Sources (1972)

No. 27. Sampling Applications of the 1970 Census Publication, Maps, and Public Use Summary Files (1972)

No. 28. From the Old to the New 1972 SIC for Establishments (1972)

No. 29. Scientific and Technological Development Activities of the Census (1973)

No. 30. Census County Division, Past and Future (1973)

No. 31. Consistency of Reporting Ethnic Origin in the Current Population Survey (1974)

No. 32. Standards for Discussion and Presentation of Errors in Data (1974)

No. 33. Characteristics of Persons in Engineering and Scientific Occupations: 1972 (1974)

No. 34. Indexes to Survey Methodology Literature (1974)

No. 35. Family (Money) Income: 1947 to 1971: Summarizing Twenty-Five Years of a Summary Statistic (1974)

No. 36. Census Bureau Programs to Measure Consumer Purchase Expectations: 1959-1973 (1974)

No. 37. The Census Bureau: A Numerator and Denominator for Measuring Change (1975)

No. 38. Comparison of Persons of Spanish Surname and Persons of Spanish Origin in the United States (1975)

No. 39. Guide for Local Area Population Projections (1977)

No. 40. The Current Population Survey: Design and Methodology (1978)

No. 41. An Evaluation of 1970 Census Occupational Classification (1978)

No. 42. 1976 Survey of Institutionalized Persons: Methods and Procedures (1978)

No. 43. Graphic Presentation of Statistical Information (1978)

No. 44. Standard Statistical Establishment Program (1978)

No. 45. The 1972-73 U.S. Consumer Expenditure Survey: A Preliminary Evaluation (1978)

No. 46. Reevaluation of the 1972-73 U.S. Consumer Expenditure Survey: A Further Examination Based on Revised Estimates of Personal Consumer Expenditures.

No. 47. Raw materials in the United States Economy: 1900-1977 (previous issues published in Working Paper series)

No. 48. Topological Principles in Cartography